2005

SPECTRA: Secure Power Efficient Clustered Topology Routing Algorithm

Waqaas Siddiqui

SPECTRA: Secure Power Efficient Clustered Topology Routing Algorithm

By

Waqaas Siddiqui

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Dr. Fei Hu
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
May, 2005.

Approved By:

Dr. Fei Hu
Primary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Muhammad Shaaban
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Greg Semeraro
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: SPECTRA: Secure Power Efficient Clustered Topology Routing Algorithm

I, Waqaas Siddiqui, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Waqaas Siddiqui

Waqaas Siddiqui

Date

Dedication

I would like to thank my parents, Mohammad and Rizwana Siddiqui, my two sisters, Fareeha and Sohaira Siddiqui, Humaira Ahmed, and the rest of the family. I would also like to thank my close personal friends, Vernon and Desiree Hosannah, and Victor and Stephanie Flagg-Rhett. Their support, encouragement, and love have been greatly appreciated.

Acknowledgements

There are a number of people I would like to thank for their help and contribution to this work. I would like to thank my advisor, Dr. Fei Hu, for his time and help. I would also like to thank my committee members, Dr, Muhammad Shaaban and Dr. Greg Semeraro.

I would also like to thank the following people:

- Chris Wells for his assistance with the simulator and in answering questions related to simulation programming.
- Richard Tolleson for his help in computer related aid
- Spencer LaDow for his help in Matlab topology construction and answering questions related to.
- Becky LaDow for proof-reading and editing
- Fareeha Siddiqui for proof reading and editing

Abstract

Wireless Sensor Networks (WSNs) have emerged as one of the hottest fields today due to their low-cost, self-organizing behavior, sensing ability in harsh environments, and their large application scope. One of the most challenging topics in WSNs is security. In some applications it is critical to provide confidentiality and authentication in order to prevent information from being compromised. However, providing key management for confidentiality and authentication is difficult due to the ad hoc nature, intermittent connectivity, and resource limitations of the network. Though traditional public key-based security protocols do exist, they need large memory bandwidths and complex algorithms, and are thus unsuitable for WSNs.

Current solutions to the security issue in WSNs were created with only authentication and confidentiality in mind. This is far from optimal, because routing and security are closely correlated. Routing and security are alike because similar steps are taken in order to achieve these functions within a given network. Therefore, security and routing can be combined together in a cross-layer design, reducing the consumption of resources.

The focus of this work is on the integration of routing and key management to provide an energy efficient security and routing solution. Towards this goal, this work proposes a security protocol that encompasses the following features: integration of security and routing, dynamic security, robust re-keying, low-complexity, and dual levels of encryption. This work combines all the robust features of current security implementations while adding additional features like dual layer encryption, resulting in an extremely efficient security protocol.

Table of Contents

List of Figures

List of Equations

List of Tables

Glossary

Wireline:	Wired networks such as LANs.
WSN:	Wireless Sensor Network (utilizing micro-sensors).
RSS:	Received Signal Strength.
SPECTRA:	Secure Power Efficient Clustered Topology Routing Algorithm
ACK:	Acknowledgement
NAK:	Not Acknowledged
JiST:	Java in Simulation Time
SWANS:	Scalable Wireless Ad-hoc Network Simulator
Clusterhead:	node with additional power (supernode) [30]
CH:	Clusterhead
BS:	Base-station
RREQ:	Route Request Message
RREP:	Route Reply Message
Weak Freshness:	provides partial message ordering, but carries no delay information

Chapter 1. Introduction

Within the last decade, wireless technology has substantially increased its presence in the marketplace. This phenomenal growth stimulated advancements in the wireless sector, eventually leading to the creation of Wireless Sensor Networks (WSNs). The primary purpose of a WSN is to gather data from a remote location in a secure fashion. WSNs are one of the hottest fields due to their low cost, high flexibility, and self organizing behavior. However, due to the nature of wireless communication, data is available in the air for any third party to acquire. This feature along with the ad hoc nature, intermittent connectivity, and resource limitations of WSNs result in a number of design challenges.

In particular, the availability of data to third parties causes numerous security issues. Therefore, it is critical to provide confidentiality and authentication in order to prevent information from being compromised. Traditionally, security is provided through public-key based protocols. However, these protocols require large memory bandwidth and complex algorithms [45]. The limited resources of WSNs make this type of security unsuitable for implementation. In response, security protocols that provide security while taking into account the unique features and resource limitations of WSNs are preferred.

Currently, there exists very little work in the security of WSNs. Some pioneering work has been done on key management/pre-distribution/generation in WSNs, mainly ad hoc networks [31, 45, 51]. But there is a common shortcoming for those schemes: they focus purely on security and ignore the particular features of WSNs [30]. These schemes implement security at the expense of energy and routing efficiency [30]. This is far from optimal, because routing and security are closely correlated. Routing and security are

alike because similar steps are taken in order to achieve these functions within a given network. Therefore, security and routing can be combined together in a cross-layered design, reducing the complexity of the protocol.

The focus of this work is on the integration of routing and key management to provide an energy efficient security and routing solution. This work combines all the robust aspects of current security protocols while adding additional features. The features comprised in this work are as follows: integration of security and routing, dynamic security, robust re-keying, low-complexity, and dual levels of encryption.

The integration of routing and security result in a cross-layered design. This is highly beneficial because it reduces redundancy and overhead by integrating similar aspects of these layers. Maintaining the idea of layers allows for a level of abstraction resulting in easy modifications and insertions into other applications. Along with an independent and integrated layer, it is important to make security dynamic to support topology changes in a WSN.

Dynamic topology is native to a WSN because nodes can fail or be added. This feature of WSNs makes it vital to have a dynamic security protocol. A dynamic security protocol adapts to network changes and can distinguish between legitimate node addition and enemy infiltration. One of the key features in supporting dynamic security is the ability to re-key.

Re-keying is a feature that is extremely valuable to WSN security [45]. The dynamic nature of WSN topology makes it vital to refresh (re-key) certain keys. This is done in order to authenticate valid nodes and to ensure that enemies do not acquire keys

easily. Without re-keying, enemies can acquire network keys through failed nodes. It is important to provide features like these without adding to protocol complexity [42].

Low-complex security is essential to WSNs because of the limited resources available in individual nodes. Therefore, traditional public key based security is unsuitable for WSNs [30]. This work uses a symmetric key based scheme in order to reduce memory requirements and computational overhead related to decrypting, encrypting, and key generation. There is always a tradeoff between changing from public key based encryption to symmetric key based encryption. Features like shorter key lengths and less complex key generation result in a less resource hungry solution but an overall reduced level of security.

A dual level of encryption is introduced in this work in order to increase the level of security. This is achieved through the use of two symmetric keys simultaneously. The level of security is increased because two separate keys need to be acquired instead of a single key. This feature combined with symmetric key encryption, results in a robust security protocol that is ideal for WSNs.

In order to design a security protocol for WSNs, an in depth understanding of the following is needed: the features of a WSN, how they operate, and current security protocols. An in depth discussion of these topics is provided in Chapter 2. Chapter 2 steps through the background of a WSN, the challenges of designing a WSN security protocol, and the shortcomings of existing work.

Following Chapter 2, Chapters 3 through 8 discuss various aspects of the proposed protocol. Chapter 3 focuses on the messages and components of the protocol architecture. It explains every type of message that is used within the system and describes all

components and their function. In Chapter 4 the steps and procedures taken by the system at initial startup are presented. It provides a detailed description of what goes on during *initial system setup*. Chapter 5 focuses on how the system functions after *initial system setup* and the aspect of security once setup has completed. Chapter 6 is dedicated to the security of the system, both during *initial system setup* and post initial setup. In Chapter 7 the protocol's ability to support secure post deployment expansion is discussed. Chapter 8 explains the simulation tool that was used to simulate the protcol and all acquired results. An in depth performance analysis of the acquired results is provided in Chapter 9. The thesis ends with conclusions and possible future improvements in Chapter 10.

Chapter 2. Background

In recent years, there has been a vast increase in the use of wireless devices as well as the emergence of a variety of different wireless technologies such as Bluetooth [39], Wireless LAN [40], Third-generation cellular networks [41], and so on. In fact, it was predicted that wireless data access will exceed wireline by the year 2004, as was seen at the end of 2004 [32] (see Figure 1).

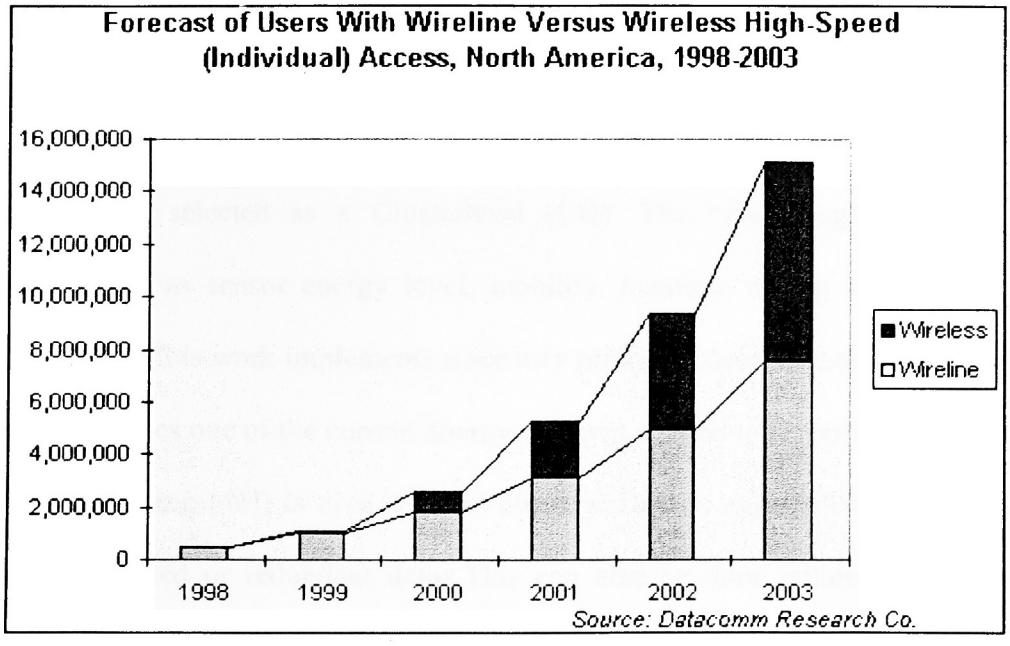

Figure 1: Wireless Data access [32]

A WSN is comprised of wireless sensor modules, called nodes. Each node is made up of a few key components: a micro-sensor to detect the desired event; a low-cost application-specific microprocessor; memory to store information; a battery for power; and a transceiver for communication between the node and the rest of the network. Micro-sensor nodes consist of cost effective and extremely power efficient parts with only the required functionality.

Owing to the small size of the sensor on each node, the sensing range of a single node is extremely limited (typically 10 to 50 meters [30]). The small size of the sensor, and the lack of large and complicated signal conditioning circuitry, leads to high data error rates ($> 10^{-3}$) [45]. However, due to their low cost nature, hundreds and thousands of nodes can be used to sense events in buildings, battlefields, or other places. In a WSN, a single event can be detected by multiple nodes. This data redundancy can be used to achieve fault tolerance (that is, nodes can fail without effect to the network).

Typically, a WSN will self-organize itself into a "cluster" architecture [46] after sensor deployment. A "cluster" includes a group of neighboring nodes where one of the group nodes is selected as a Clusterhead (CH). The "clustering" algorithms use parameters such as sensor energy level, mobility, location, etc. to form clusters and determine CHs. This work implements a security protocol based on a clustered topology. This work assumes one of the current energy efficient clustering algorithms (such as load balancing clustering [46]) is used to form clusters. Data is aggregated by the CH who removes duplicated or redundant data. This can also be done within the network by having nodes closer to the CH aggregate the data coming from nodes farther away through eavesdropping (see section 5.3).

In terms of wireless sensor design, a paradigm shift occurred from traditional macro-sensors to the micro-sensors used in Wireless Sensor Networks (WSNs). Sensors are becoming smaller and cost effective. Traditionally, automated sensors, called macro-sensors, are large (size > 20mm) and consist of complex functionality [43]. Macro-sensors generally have a large battery or a reliable power source, resulting in a long lifetime (at least years [43]). Macro-sensor networks normally consist of a small number

of sensors (typically tens of), due to the high cost of each sensor (typically > $100 each sensor) [43]. Advances in technology have lead to a miniaturization of circuits (smaller transistors and more layers), power efficient transceivers and microprocessors, and more effective RF transmission. These advances enabled sensors to become extremely small, more power efficient, and more cost effective; for instance, a UC Berkeley "smart dust" can work for a year on two AA batteries, and is about the size of a penny. These are called micro-sensors [44]. Micro-sensors are cheap and expendable (typically < $10 [45]), allowing them to be deployed in a single large-scale network.

WSNs based on micro-sensors[1] make possible the reliable monitoring of various environments for different applications. Many new features are provided to sensing applications by WSNs. These include:

- **Fault Tolerance**: WSNs are more fault tolerant than traditional macro-sensor networks because sensors are close enough together that data is correlated [30].
- **Improved Accuracy**: An individual micro-sensor generates less accurate data then an individual macro-sensor. However, because nodes are close together and their data is redundant, once aggregated, the data enhances the accuracy of the sensed signal and reduces the uncorrelated noise [45].
- **Low Cost**: Even though a large number of micro-sensors nodes are used, these nodes are smaller, less reliable, less complex, less accurate individually, and therefore cheaper [30].
- **Large range of sensing**: Even though individual sensor nodes have limited wireless communication ranges (< 15 meters [30]), sensors can relay information

[1] In this work, only micro-sensor based WSNs are mentioned.

in a multi-hop path. Therefore, sensor data can be transmitted longer distances (e.g. a few miles).

The first successful micro-sensor products were termed '*Smart Dust*,' or '*Motes*', which are now provided by Crossbow Company [33]. There are over 30 integrated sensor network providers such as Crossbow [33], Millennial Net [35], Ember Corporation [36], Sensicast Systems [37], MicroStrain [38], to name a few. These products have been integrated into various applications such as: environmental, defense, aerospace, etc.

This work focuses on proving that an integrated approach to routing and security reduces energy consumption. Existing work in WSN Security looks at security through the application layer, disregarding the internal architecture of a WSN. Only end to end security is of concern (as seen in Figure 2), not internal security. If an integrated approach is adopted, security can be insured at every step of the internal network including the ends.

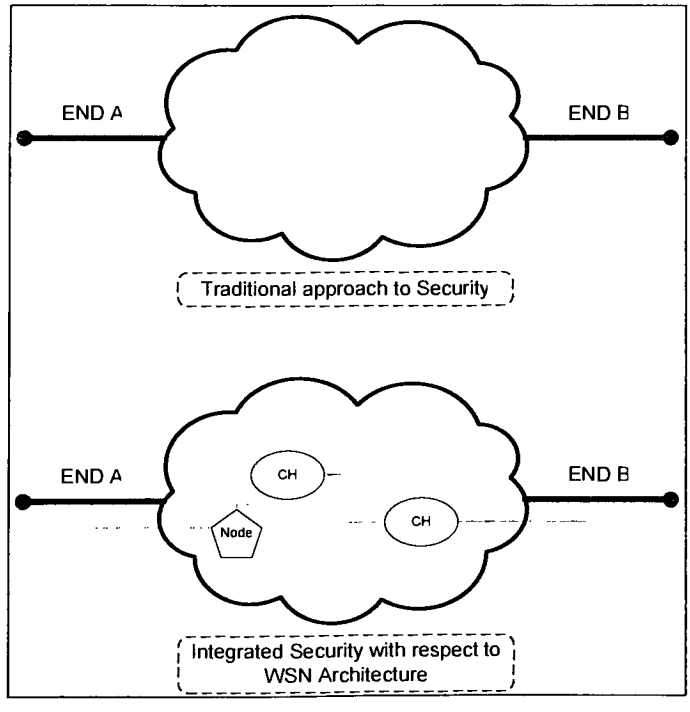

Figure 2: Integrated security approach versus traditional approach

This work shows the energy benefits of this approach through simulations. An analytical mathematical model is beyond the scope of this work. In addition, the lack of current work in this particular approach makes it difficult to provide a comparative analysis. Therefore, this work focuses on showing the energy improvements of this approach with respect to traditional approaches.

2.1 Challenges of Designing Sensor Network Security

Many of the rising applications, such as military and medical applications require security to insure data confidentiality, owing to the sensitive nature of the data. Unlike a wired network, wireless networks broadcast all their messages over a radio link, and these messages can therefore be picked up by unauthorized devices. The focus of this work is on the integration of routing and key management to provide an energy efficient security and routing solution. A lightweight security protocol based on a hierarchical WSN routing protocol is designed.

An important challenge in the implementation of security in WSNs is that two important resources—memory and energy—are extremely limited due to the size of the sensor, which can be as small as a penny [30]. Conventional security schemes such as Kerberos [47], Needham-Schroeder [48], Otway-Rees [49], Bellare-Rogaway [50] require extremely large amounts of memory (< 100's of MBs) [6] and are computationally intensive. Thus, sensor network security schemes should not consume copious amounts of energy (< 10's of Joules) [30] during key management and should not use large amounts of memory (< 100's of MBs) [30] to store security information. It is also vital to design security schemes such that they are adaptive to network topology

changes due to node failure, node addition, and interruptions in the wireless transmission medium [30]. For example, if nodes fail due to low power, messages will fail to be delivered because routes containing the dead nodes still exist. In the case of node addition, it is important to distinguish between legitimate sensor traffic, and the infiltration of the network by an enemy node. Also, intermittent connectivity must be considered because the security scheme should be able to deal with wireless errors. These considerations along with the resource constraints imposed make the design of a security scheme a challenging task.

This work recognizes these issues and focuses on designing a security protocol, called SPECTRA (Secure Power Efficient Clustered Topology Algorithm), which is energy efficient (less than 10's of mJs) [42] and exploits the synergy between adjacent layers of the protocol stack. A cross-layered approach to the design of a security protocol can reduce computation, communication, and memory requirements [42]. However, in conventional WSN security schemes such as LEAP [25] and μTESLA [9], Routing and Security are performed separately, leading to superfluous communication between sensors. In contrast, the results of this work show that a cross-layer design would eliminate this redundant communication by performing both functions in a single exchange of messages.

2.2 Current work on sensor network security

Currently, the existing work in the security of WSNs is very limited. Some of the key attributes of prominent research work in WSN security is as follows: broadcast

authentication, key predistribution, and intrusion detection. These features are discussed in greater detail below.

Broadcast authentication: The pioneering work on securing sensor networks is SPINS [6,7], which proposes μTESLA, an important innovation for achieving broadcast authentication of any messages sent from the base-station (BS). An improved multi-level μTESLA key-chain mechanism was proposed in [8,9].

Key predistribution: to make sure that any pair of sensors can find a shared key to encrypt/decrypt their messages. Note that the use of a public key (asymmetric) scheme is not plausible, since the limited memory of a sensor may not be large enough to hold a typical public key of a few thousand bytes [6]. A trusted-server scheme is not advised, since in WSNs, one generally cannot assume any trusted infrastructure [11]. Instead, it is better suited to bootstrap secure communications among sensors using key predistribution to preload some symmetric keys within the sensors. A key-pool scheme was suggested in [12] to guarantee that any two nodes share at least one pairwise key with a certain probability. Multiple pairwise keys may be found between nodes through the schemes in [13]. Key predistribution schemes utilizing location information were described in [14-16].

Intrusion detection: In WSNs, a challenging problem is to detect compromised sensors and then to tolerate intrusions by bypassing the malicious sensors [17]. A basic requirement is to limit the impact of compromised sensors into a limited local area instead of allowing an intrusion to affect the majority of the network [18,19]. The BS may be responsible for most of the intrusion detection since the sensors are too limited in memory and energy [20].

SPECTRA combines a few key features from existing protocols. SPINS [7] and Pebblenets [4] inspired the utilization of global authentication and a clustered topology in the SPECTRA protocol. In the following discussion, these two security protocols are explained.

2.2.1 SPINS

A prominent security protocol in WSNs is SPINS [7], which was designed and developed at UC Berkeley. SPINS was implemented using two already existing security protocols. It uses SNEP[7] and μTESLA[7] as its two building blocks. SNEP was used to provide data confidentiality, two-party data authentication, integrity, and freshness [7]. μTESLA was used to provide authentication for broadcast purposes (Figure 3 shows a diagram of the time released function of μTESLA to achieve authentication).

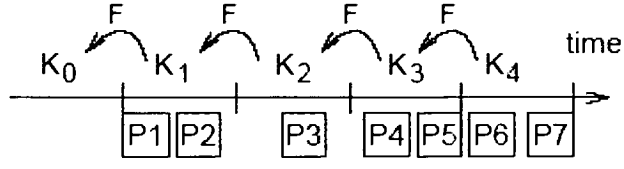

Figure 3: μTESLA time released key chain for source authentication [9]

SNEP provides a number of advantages like semantic security, data authentication, replay protection, weak freshness, and a low communication overhead. These features make SNEP an extremely robust protocol and a useful security block for implementation. μTESLA is not a security protocol like SNEP but an authentication protocol. μTESLA is a derivative of the TESLA [7] protocol and is specifically designed for sensor networks. The original TESLA protocol used symmetric key encryption, which

is computationally too intensive for sensor networks [9]. μTESLA implements a one-way time released key chain that is used for source authentication. This key is refreshed periodically, resulting in global authentication [9]. This idea of global authentication is also implemented in SPECTRA but without the use of a key chain.

SPINS is a powerful protocol because of its periodic global authentication and flexible scalability, yet the weakness of SPINS exists in network topology and global authentication. SPINS does not implement clusters and instead has each node share a pair-wise key with the BS. This is inefficient due to the communication overhead of having each node communicate directly with the BS. This results in a lowered system lifetime, the loss of data aggregation, and the occurrence of data duplication. Global authentication using μTESLA suffers from the same problem of node-to-BS communication.

Like SPINS, SPECTRA implements global authentication but without node-to-BS communication. SPECTRA supports clustering which results in data aggregation, the reduction of data duplication, and a longer system lifetime. Aspects of SPECTRA are extended upon some of the robust features of SPINS. However, this work modifies these features to support system lifetime, data aggregation, and a cross-layered design (which is explored in Chapter 5).

2.2.2 Pebblenets

Pebblenets [4] is a security protocol that utilizes clusters and multiple keys. It implements global authentication through a periodically refreshed traffic encryption key (TEK). Every node within the system is authenticated through this global TEK.

Pebblenets also implements a network backbone that is comprised of all head cluster nodes. A unique key is used for intra-backbone communication while a different unique key is used for intra-cluster communication. The Pebblenets protocol is comprised of two main phases. In the first phase, the network nodes organize into clusters and create a backbone. The second phase consists of a small number of the backbone nodes generating new keys, which are then broadcast to all nodes [4]. The following diagrams depict a Pebblenet showing its group of clusters and its cluster backbone.

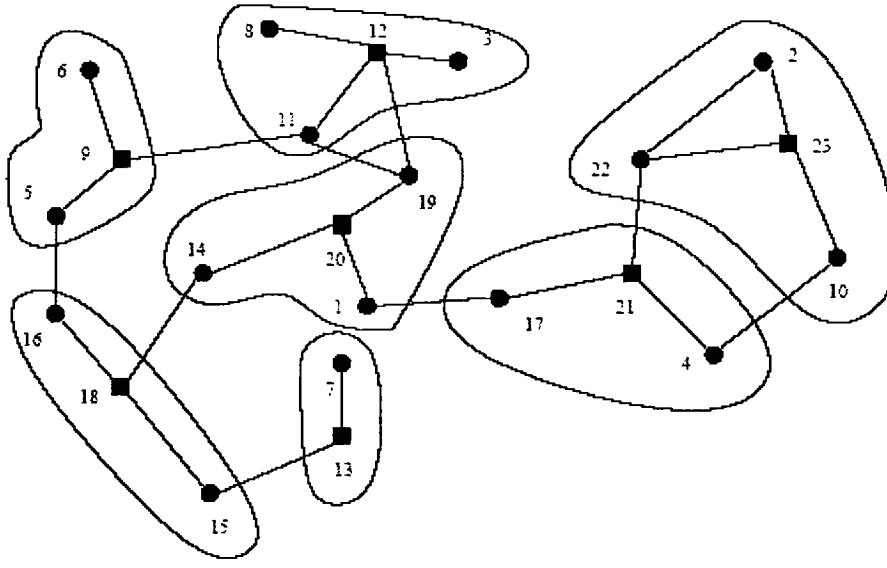

Figure 4: Pebblenet partitioned into clusters [4]

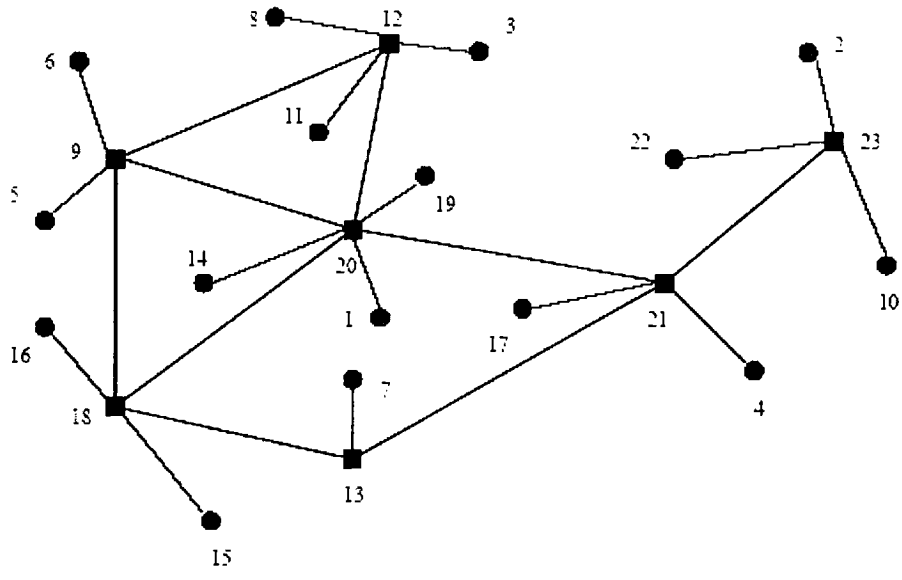

Figure 5: Pebblenet cluster backbone [4]

There are many similarities between Pebblenets and SPECTRA, such as the utilization of clustering, a network backbone, global authentication, and multiple keys. However, unlike SPECTRA, Pebblenets does not implement a routing protocol along with its security protocol [4]. A downfall to Pebblenets is the idea of having all keys being generated by a backbone node. Key generation is based upon a nodes *personal key* that is provided at deployment. Therefore, if a priority node is captured, new keys can be generated and distributed, resulting in full system compromise. Due to Pebblenets lacking a routing protocol, network and security setup becomes complicated (requires established routing before adding security) and energy inefficient. SPECTRA solves these problems by implementing a routing protocol and having a secure source generate the *system key* (to be discussed in section 3.7.4).

2.3 Summary

This chapter discussed WSNs, how they work, their features, and the challenges of designing security protocols in WSNs. The design considerations and associated challenges pertaining to WSN security were discussed methodically in order to understand the motivations behind the novel approach to security proposed in this work. This chapter also provided information on existing work in the field of WSN security, including the robust features and shortcomings of current work. Now that the motivations and challenges behind WSN security are better understood, the architecture of the proposed protocol in this work can be more easily comprehended. The following chapter steps through the architecture of the proposed protocol and the components within it.

Chapter 3. Proposed Protocol Architecture and Components

In response to the limitations (see Chapter 2) of contemporary security protocols for WSNs, this work proposes a new WSN protocol named SPECTRA that emphasizes security without sacrificing power-efficiency. SPECTRA was designed to take into account the unique requirements of a WSN, such as low-power, low-memory, limited computation, and the error-prone wireless radio medium.

There are many variables that can be considered in the area of wireless sensor networks. However, many of these variables are only in question in certain layers of the protocol stack. With this in mind, certain assumptions are made when dealing with certain layers in order to analyze only the variables that pertain to the layer in question. It is important to understand the assumptions that are made for the area of security. The following assumptions are generally accepted in security considerations for WSNs.

- **Base-Station**: the base station is assumed to be trusted and cannot be compromised. It also has unlimited transmission power and a redundant power source. The base station is responsible for re-keying and refreshing the network, as well as collecting data from the entire network.
- **Wireless errors**: wireless errors are dealt within the physical layer of the protocol stack and outside the scope of this work.
- **Wireless bottlenecks**: bottlenecks and traffic congestion are dealt with in the physical layer of the protocol stack and outside the scope of this work.
- **Frequency of Communication**: this characteristic is dealt within the application layer and outside the scope of this work. The application layer is responsible for generating and periodically sending data.

- **Predistribution scheme**: the initial routing structure, or predistribution structures are assumed to be in place. Therefore, initial authentication is conducted by utilizing the already existing topology. Therefore, nodes do not have to directly communicate with the base-station. Though this work supports predistribution, it deals strictly with post distribution security.
- **Data aggregation**: implemented through a data aggregation policy that is integrated into this work. Data aggregation itself is outside the scope of this work and simple integrated.

In order to understand how the SPECTRA protocol works in its entirety, a thorough understanding of each component utilized in the architecture is needed. This chapter will discuss the main functions of different components in SPECTRA and the innovations behind this protocol. Components include physical objects in the network along with specific entities that are used to provide network functions (such as keys, messages, and route formation). A system diagram that provides an overview of the SPECTRA protocol can be seen in Figure 6. The diagram provides an overall picture of the SPECTRA protocol and the components that comprise it: clusters, nodes, CHs, a BS, and messages. This chapter will provide an in depth coverage of all messages used within the SPECTRA network along with the types of keys and how they are used. Routing, on the other hand, will only be covered briefly because routing takes place during *initial system setup* (see Chapter 4).

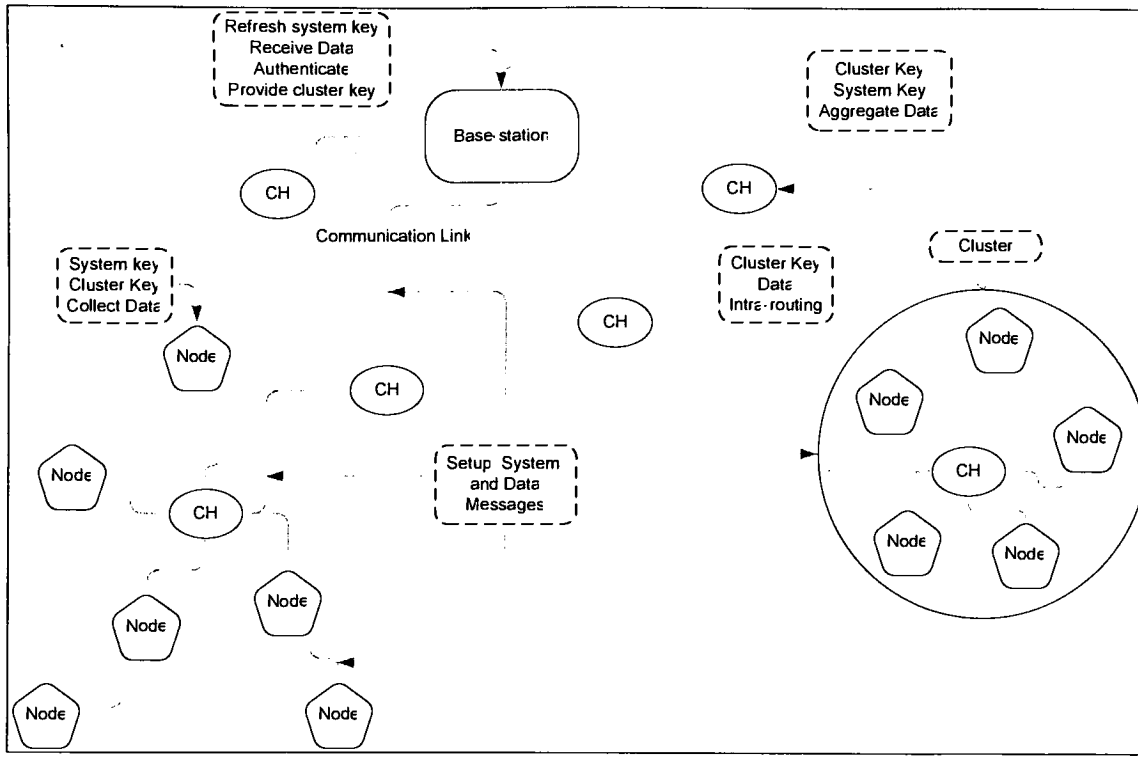

Figure 6: SPECTRA Overview Diagram

3.1 Innovations of Proposed Protocol

SPECTRA is innovative because it combines the robust features of existing protocols while adding additional features. The features comprised in this work are as follows: integration of security and routing, dynamic security, robust re-keying, low-complexity, and dual levels of encryption. These features are discussed in greater detail below.

(1) Seamless integration of security with scalable routing protocols: A cross-layered design approach is highly beneficial because it reduces redundancy and overhead by integrating related layers [42]. SPECTRA is highly practical because it was designed to integrate routing and security without sacrificing power. It is a dynamic protocol where security is provided independent of central control. Existing work overlooks the idea that

any security scheme should be seamlessly integrated with the special characteristics of WSN architecture, especially routing protocols; otherwise, the security scheme may not be practical or energy-efficient from the network protocol point of view [30].

Thorough research of the field has found that most of the existing WSN security strategies focus only on key management / security algorithms. For example, all existing key-predistribution schemes try to establish pairwise keys between each pair of nodes. However, most sensors do not need to establish a direct point-to-point secure channel with sensors multiple hops away since WSNs use hop-to-hop communication techniques to achieve long distance transmission. One of the most famous schemes, SPINS [7], simply assumes a flooding-based, spanning-tree architecture with the BS as the tree-root. However, the establishment and maintenance of a global spanning tree in a large-scale WSN with a large footprint may not only bring unacceptable communication overhead (and thus increased energy consumption[2]), but also cause a large transmission delay and make the impractical assumption of time synchronization in μTESLA (a broadcast authentication protocol [9]). An important feature of this work is that it has a robust hop-to-hop transmission scheme and can recover from multiple key losses. Routing and security can be combined because similar steps are taken in order to achieve these functions within a given network.

(2) Dynamic Security: Dynamic network topology is native to WSNs because nodes can fail or be added. This feature requires a security protocol to be adaptive to changes in network topology. Topology changes occur due to node fall out or node addition. In the case where nodes fall out, these nodes must be removed from the network

[2] Most of the sensor's power is consumed by communication, not instruction processing [30]. The energy used to communicate one bit of data can be used to execute over 1000 local instructions [6]. High energy consumption can drain sensors quickly and thus shorten the network lifetime.

completely. If this is not done, these nodes can be impersonated by an enemy because the nodes still exist within the network. If security is not updated, keys can be acquired from nodes that have failed. In the case of node addition, a protocol must be able to distinguish between legitimate node addition and attempted enemy infiltration. Given these reasons, adaptive security should be present for WSN applications in order to ensure overall network security. If adaptive security protocols aren't designed and employed, network compromise becomes far easier.

(3) Robust re-keying: As mentioned above, a dominant feature of WSNs is the dynamic network topology. Sensors may run out of battery power and new sensors may be added to the network. More importantly, network enemies might compromise sensors and all security information in those sensors may be obtained. Therefore, after key-predistribution and sensor deployment, a re-keying scheme should be used to update some types of keys such as group keys and session keys. This is done to ensure that enemies cannot acquire the keys easily. In this work, a re-keying protocol that can adapt to dynamic network conditions such as sensor compromise and topology change was implemented.

(4) Low-complex implementation. This work uses a symmetric-key-based scheme because memory use is a major concern in WSNs [30]. This prohibits the use of memory-intensive asymmetric keying schemes. Asymmetric keying schemes need more complex cryptography calculations and protocols, which can bring too much communication overhead compared to symmetric-key-based schemes. This protocol also has low transmission energy due to its cluster-based key management. Key generation and distribution protocols with several hash computations and global key delivery rounds can

quickly drain the limited power of a node. Due to WSNs consisting of hundreds, if not thousands, of nodes, network topology/densities can change; therefore, a centralized or flooding-based security scheme cannot scale well. Thus distributed algorithms and localized coordination to achieve global convergence and scalability is preferred [30].

(5) Dual levels of Encryption: In switching from traditional public key based protocols to symmetric key based protocols, overall security is reduced. It is harder to compromise public keys because they are generated from more complex algorithms and are far larger. Symmetric keys, on the other hand, are shorter and are generated through far simpler functions, making compromise all the simpler. SPECTRA attempts to increase the level of security by using two symmetric keys simultaneously. This requires an enemy to compromise two separate keys instead of a single key.

In summary, SPECTRA combines the following features into a protocol catered for WSN security:

- Cross layered design approach to security and routing
- Dynamic security
- Robust re-keying
- Low complexity
- Dual levels of encryption
- Global Authentication to support global eavesdropping
- Separate Routing and Datagram encryption
- Maintenance of clusters without adding CHs

As mentioned earlier in Chapter 2, SPECTRA combines features from other protocols. Global Authentication is a feature that is loosely inspired by the SPINS

protocol [7], but its implementation in SPECTRA is significantly different. SPINS proposes global authentication to take place through direct communication between the nodes and the base station (BS). SPECTRA provides global authentication through a broadcast approach, reducing the overhead corresponding to direct transmission to the BS. Transmission overhead is also decreased through the cluster based topology employed by the SPECTRA protocol.

SPECTRA achieves power-efficiency by reducing inter-node communication overhead through data aggregation and efficient routing via a wireless backbone consisting of CHs (explored in Chapter 5). Figure 6 shows a brief overview of the protocol architecture and its physical components: nodes, CHs, and clusters. The nodes organize themselves into clusters around a CH. A CH is similar to a normal sensor node in all respects, except that it has a larger battery leading to a longer life. Network topology or clustering is conducted to optimize CH distribution (see 3.5).

CHs provide security and routing services to all the nodes in the network. Other cluster topology protocols (such as LEACH [51], Pebblenets [4], LiSP [31]) do not define the notion of a CH with a larger battery. This deficiency places disproportionate power demands on their CHs (regular sensor nodes), resulting in earlier node failure and reduced system lifetime [4]. The power requirements for a CH are larger because it handles all intra-cluster and inter-cluster communication. CHs aggregate all the data from the sensor nodes in their cluster and transmit it to the BS all at once, reducing the total number of messages sent. Before messages are transmitted, keys are used within the network to encrypt them.

SPECTRA uses three types of symmetric keys: a *Personal key*, a *Cluster key*, and a *System key*. *Initial system setup* utilizes the *initial key* and the *personal key* to encrypt its messages, insuring confidentiality and authentication during the normally-vulnerable *initial system setup phase*. This approach also insures confidentiality of the network topology, and authentication along with every message sent within the system (discussed in Chapter 6). Therefore, unlike other proposed protocols such as SPINS [7], LEACH [51], and Pebblenets [4] (see section 2.2), authentication does not need to be performed separately, and is integrated into routing.

3.2 Role of Base-station in SPECTRA

In a WSN, a BS collects results from all sensors [30]. Current WSN security research assumes the BS to be invulnerable to compromise, and is always trusted [30]. The BS is also assumed to have effectively unlimited battery power (such as a redundant AC power source), wireless transmission range, memory space, and computational capacity. It shares a pair-wise key with every node and CH in the network. The BS issues *cluster key*s to authenticated CHs during the *cluster organization phase*. It is also responsible for periodically broadcasting the latest *system key* to the network. The functions of the BS can be summarized as follows:

- Initial authentication of nodes and CHs during *initial system setup*
- Global authentication of a system once it has been set up
- Data querying (requesting data from the network)
- Collect information regarding network topology from CHs
- Authentication of new nodes added after initial setup

The BS's primary function during the *initial system setup phase* is to authenticate every node and CH in the system. This is accomplished by an exchange of messages between the nodes and the BS. Once all nodes and CHs are authenticated in the *initial system setup phase*, the *cluster-organization phase* begins. The BS's role in this phase is to respond to requests (by authenticated CHs) for cluster-keys.

Once *system setup* and *cluster-organization* phases are completed, the BSs role is simplified to generating data queries in response to user requests, and periodically refreshing the *system key*. The *system key* is broadcast every epoch (*system key* expiration period), encrypted with the old *system key*. Nodes could fail to get the latest *system key* due to wireless transmission errors. SPECTRA overcomes this challenge by broadcasting the new *system key* three times in rapid succession at the start of every epoch. This also provides periodic authentication of every node and CH in the network (discussed in 4.1 and 6.2).

The BS also keeps track of network topology. This includes knowledge of all CHs, their clusters and *cluster keys*, and the member nodes in each cluster. This knowledge is used in the event of CH compromise (see section 6.6). In this situation, the BS is able to intelligently reorganize the cluster's member nodes into a new cluster.

3.3 Role of Sensor Nodes in SPECTRA

Nodes are simple, small, and cheap [30]. Their limited functionality includes sensing and generating data, which they then transmit to their CH. Within a cluster, nodes organize themselves into multi-hop routes to the CH [46].

Nodes are periodically authenticated by receiving and decrypting the latest *system key*. They use this key to encrypt their routing headers so that other nodes will not disregard their messages. In theory, all nodes along the route know the latest *system key*, enabling them to determine the next hop from the routing header. This is achieved by decrypting the packet using the latest *system key* in order to obtain the routing header. The routing header is then used to determine the next hop.

After authentication, all nodes belonging to the same cluster get a shared *cluster key* from their CH which is used to encrypt the data portions of their messages. This *cluster key* allows for eavesdropping to prevent data duplication (see section 5.3).

3.4 Role of Clusterheads in SPECTRA

CHs are nodes with additional responsibilities. During *system setup*, CHs organize nearby nodes into a cluster. In the situation where nodes are equally placed from multiple CHs, nodes join the CH that responded first. They are also responsible for acquiring and issuing the cluster-key to the nodes in their cluster. Each CH obtains a *cluster key* from the BS after the *initial system setup phase*. The CH transmits the *cluster key* to nearby nodes in response to their requests to join the cluster, encrypted with the *system key*.

CHs' main function within the cluster is to aggregate data received from cluster member nodes. Data is collected and duplicate data is discarded, resulting in the aggregation of all member node data. Aggregated data can be forwarded to the BS in response to a data query. All CHs together form a wireless routing backbone.

3.5 Role of Clusters

Clusters are formed by using an existing energy efficient clustering algorithm. SPECTRA incorporates the load balancing clustering algorithm, which can form clusters with balanced traffic distribution among clusters [46]. Clusters are beneficial in WSNs because they reduce the distance of communication required for nodes within the network [4, 30], i.e. every node does not have to directly communicate with the BS, increasing system lifetime.

Cluster formation begins with an authenticated CH requesting a *cluster key* from the BS. The CH receives the new *cluster key* and decrypts it with the *system key*. After the CH has the *cluster key*, it broadcasts an advertisement to all nearby nodes. Nodes listen to these advertisements, and after waiting for some period of time (half an epoch see 6.2.2.1), join the cluster from which they receive the highest RSS (Received Signal Strength). The request of joining the cluster is encrypted with the *system key*. The CH responds by sending the *cluster key* back to the node, encrypted with the *system key*. The CH is responsible for keeping track of all member nodes in its cluster. After the *cluster organization phase* is completed, the CH forwards a list of all nodes in the cluster (through a message) to the BS, encrypted with the *system key*.

The data portions of all messages sent within the cluster are encrypted with the *cluster key*. Like all messages in the SPECTRA network, the routing headers of messages sent within the cluster are all encrypted with the *system key*. This dual-key system insures both authentication, and confidentiality. The usage of two different keys (cluster and *system keys*) within a cluster provides double authentication: The *system key*

authenticates a node as a member of the network, whereas the *cluster key* authenticates a node as a member of the cluster.

3.6 Routing

Clusters have an internal topology where messages are forwarded within the cluster through multi-hop routes to the CH. These routes are organized in a multi-hop route to the CHs. During the *cluster-organization phase*, all nodes send out *Node Advertisements*. These advertisements make all nodes aware of their neighbors, and allow for the construction of Intra-Cluster and Inter-Cluster routing tables.

3.6.1 Inter-Cluster Routing

Data messages generated by nodes must be collected and aggregated at the CH. Periodically or in response to a data request message, this aggregated data is forwarded to the BS. If the BS is not a neighbor of the sending CH, multi-hop routes are required to the BS. These multi-hop routes only contain CHs and do not use nodes to forward data.

Inter-cluster routing organizes multi-hop paths to the BS and only consist of CH-to-CH communication, resulting in the CH backbone of the SPECTRA network. This backbone is used for all normal system operations. These paths are determined through the use of Route Request Messages (*RREQ*) and Route Reply Messages (*RREP*) (further details of the use of *RREQ*s and *RREP*s will be discussed in section 3.8.3.3). The transfer of routing messages allows CHs to form their routing tables, which are used for intra-cluster and CH to BS communication.

3.6.2 Intra-Cluster Routing

Intra-cluster routing is used to organize multi-hop paths to the CH inside the cluster. Nodes send CHs their collected data periodically or in response to data request messages. If the node is not a neighbor of the CH, the node must determine a multi-hop route to the CH. Determining a multi-hop route entails broadcasting an *RREQ*. The *RREQ* propagates to the destination, which generates an *RREP* message. The *RREP* is returned to the sending node that adds the new route (contained in the *RREP*) to its routing table (further details of the use of *RREQ*s and *RREP*s can be seen section 3.8.3.3).

3.6.3 Intra-Cluster and Inter-Cluster Routing Tables

Routing information is contained in the routing tables. Typically, each node or CH only has knowledge of a small number of routes due to memory limitations. Routing tables are filled in through the exchange of system routing messages, a detailed description of which can be found in section 3.8.3.3.

Each route in the routing table contains the number of hops from a source node to a destination node and the addresses of all the intervening nodes in a route. Every message in the SPECTRA system must have a routing header. Except in the case of broadcast messages, the routing header always contains a route copied from the routing table of the source node.

3.7 Keys

SPECTRA achieves security within the network through the use of pair-wise keys for encryption. This section provides an overview of all keys used within SPECTRA: the

personal key, the *cluster key*, the *initial key*, and the *system key*. These keys are used to encrypt every message passed within the SPECTRA network. All keys within the SPECTRA network are computed through one-way radix hash functions (see 6.1) and a pseudo-random number generator. The pseudo random number generator is used to generate a number that is the desired key length. A one-way radix hash function is then applied to this number in order to generate the key. In the case of refreshing a current key, the current key is used in place of the generated number and the hash function is applied to the current key to generate the new key. The *personal key*s are generated prior to deployment and are stored within the node memory. The following table shows the definitions for all notations used in the key expressions:

Notation	Definition
$\underset{one-way}{\overset{f}{\rightarrow}}$	One-way radix hash function used to generate the keys
$PRNG$	Pseudo Random Number Generator used to generate the number that is the desired key length
x	The key size
K_p	*Personal Key*
$K_{Cluster}$	*Cluster Key*
$K_{current\ K_{cluster}}$	Current *cluster key* that is used to generate the refreshed key
$K_{refreshed\ Cluster}$	Refreshed *cluster key*
K_{System}	*System key*
$K_{current\ K_{System}}$	Current *system key* used to generate the refreshed key
$K_{refreshed\ System}$	Refreshed *system key*

Table 1: Symbol Definitions for Key Expressions

The following expressions depict the key generation for each type of key:

$$f_{one-way} \rightarrow (PRNG(x)) = K_p$$

Equation 1: Expression for generating *Personal keys*

$$f_{one-way} \rightarrow (PRNG(x)) = K_{Cluster}$$

Equation 2: Expression for generating *Cluster keys*

$$f_{one-way} \rightarrow (PRNG(K_{current\ K_{cluster}})) = K_{refreshed\ Cluster}$$

Equation 3: Expression for generating refreshed *Cluster keys*

$$f_{one-way} \rightarrow (PRNG(x)) = K_{System}$$

Equation 4: Expression for generating *System keys*

$$f_{one-way} \rightarrow (PRNG(K_{current\ K_{System}})) = K_{refreshed\ System}$$

Equation 5: Expression for generating refreshed *Cluster keys*

3.7.1 *Personal key*

Personal keys are used for initial authentication in the SPECTRA network. They are pair-wise keys that are only used once for authentication and are considered invalid thereafter. This is done in order to ensure that a failed node cannot be reinserted into the network by an enemy.

3.7.1.1 Clusterhead *Personal key*

CHs use their *personal key* during *initial system setup* when first authenticating with the BS. On system startup, the CHs all send a request to the BS, encrypted with their *personal* and *initial keys*. Because the *initial key* is used, the routing information for the authentication message can be eavesdropped, allowing neighbors to forward the request on. The request is for a *cluster key*, and to register them as a CH. The BS responds by sending both the latest *system key* and a new cluster-key encrypted with the personal-key of the requesting CH.

When a new CH is added to the system after the end of the initial deployment phase, a similar process is used to authenticate the new CH. Before the new CH is deployed, its *personal key* and unique node ID are registered with the BS. The new CH sends a request for a new cluster-key encrypted with its *personal key*, and the BS replies by sending the *system key* and cluster-key, encrypted with the *personal key* and *initial key* (3.7.3). This also serves to authenticate the new CHs.

3.7.1.2 Node *Personal key*

Each node has a pair-wise *personal key* shared between that node and the BS. These *personal keys* are programmed into each node before system deployment, and are used to authenticate new sensor nodes joining an existing SPECTRA network. The *personal key* is used to encrypt a message broadcast when the node attempts to join the system. This message is received by the BS, which shares the pair-wise *personal key* that was used to encrypt the message. The BS then sends out the latest *system key* to the joining node, encrypted again with the *personal key*.

Once the new node has the latest system-key, it has been authenticated to the network, and can attempt to join a cluster. Encrypted with this new *system key*, the node then broadcasts a request for advertisements from nearby clusters. All the nearby CHs respond with an advertisement, encrypted with the *system key*. This advertisement contains the cluster ID number. The node requests to join the closest cluster, based on the received signal strengths of these advertisements. The CH responds by sending its *cluster key* to the joining node, encrypted with the system-key. Finally, the CH adds the new node to its node table (a table of all registered cluster members), and notifies the BS of this addition.

3.7.2 *Cluster key*

A *cluster key* is negotiated between the CH and the BS, and is then distributed to every member of the cluster. In order to get a cluster-key, the CH must authenticate itself with the BS using its *personal key*. Within a cluster, all messages are encrypted with the *cluster key*. The notion of a *cluster key* allows nodes to eavesdrop on packets in multi-hop routes to the CH. If a node receives and decrypts a data packet that duplicates its own, it will forward the original packet, but it will not send its data.

The *cluster key* is the primary means of insuring data confidentiality. All sensor data from the nodes is encrypted with the *cluster key* before being forwarded to the CH. A second layer of protection is achieved by encrypting the routing header and the already encrypted message content with the *system key*. Both keys are needed in order to understand the contents of the message. The contents of the routing header (encrypted

with the *system key*) must be known in order to choose which key to decrypt the body of the message with (most commonly the *cluster key*).

SPECTRA supports both periodic refreshing of the *cluster key* at the CH as well as refreshing in response to compromise. The desired functionality is dependent on the level of security threat in the surrounding network. If periodic refreshing is chosen, the CH generates a new *cluster key* and broadcasts it to its cluster, encrypted with the latest *system key*. In the case of a compromise, negotiating with the BS generates an entirely new *cluster key*. All nodes are authenticated with the BS before receiving the new *cluster key*.

3.7.3 *Initial Key*

During *initial system setup*, every node and CH must authenticate themselves with the BS using the *initial key*, their *personal key*, and their node ID number. The purpose of the *initial key* is to encrypt the routing information of the initial authentication message. This ensures that all information within the network is encrypted and confidential. Authentication comes from the BS in the form of the latest *system key* which is used to replace the *initial key*.

3.7.4 *System key*

During *system setup*, every node and every CH must authenticate itself with the BS using the *initial key*, its *personal key*, and its node ID number. Authentication comes from the BS in the form of the latest *system key*. All of the routing headers of all packets in the system are encrypted with the *system key* (except for the initial *authentication*

request see 3.7.3), and so it is impossible to send or receive a packet without the latest *system key*. *System key*s expire periodically in time intervals denoted as epochs. At the start of every epoch, the BS broadcasts a new *system key* three times in rapid succession, encrypted with the previous *system key*. All nodes in the system receive this broadcast, and use the old *system key* to decrypt the new *system key*. The following sections discuss the role of the *system key* in CHs and nodes.

3.7.4.1 Nodes

Nodes only use the *system key* for authentication and encrypting routing headers. Whereas they use the *cluster key* for encrypting the data portions off their messages. Together, the *system key* and the *cluster key* achieve confidentiality.

3.7.4.2 Clusterheads

CHs use the system-key for multi-hop routing to the BS. Unlike nodes, CHs use the *system key* for encrypting both the routing header and the data portion of a message. They also use the *system key* for authentication.

3.8 SPECTRA Message Types

SPECTRA uses three types of messages: setup messages, data messages, and system messages for different functions. Setup messages are used for tasks that pertain to setting up the system, such as: node and CH addition, authentication, key refreshing. Data messages are used for sending and receiving data as well as generating data queries. System messages are generally used for tasks that affect the system or its topology, for example: node removal. A breakdown of all the messages within SPECTRA can be seen in Figure 7. Different messages are encrypted with different keys. Since different types of keys are utilized, security is enhanced because both the key type and the corresponding message type must be known.

Table 2 below shows the definition of all notations used in the message descriptions.

Notation	Definition

$K_{Cluster}$	Cluster Key
K_{System}	System key
Rt Header	Routing header for a packet. Holds the route from source to destination.
Data	Data or information that is being relayed in the message
AggData	Aggregated data from a cluster (group of nodes' data) being sent in a message to the BS.

Table 2: Symbol Definitions for Messages

The diagrams below show example of messages that are generated at the node and at the CH.

$$K_{System}\left[Rt\ \ Header \mid K_{Cluster}(Data)\right]$$

Equation 6: Message involved in node-to-node and CH-to-node communication

$$K_{System}\left[Rt\ \ Header \mid K_{System}(AggData)\right]$$

Equation 7: Message involved in CH-to-CH and CH-to-BS communication

In the case of system messages, the data portions of the messages seen above are replaced with the corresponding message. The diagram below shows a tree structure of all messages that are used in SPECTRA as well as which part of the system they pertain to. A detailed description of each message can be found later in this section (3.8.2, 3.8.3, and 3.8.4).

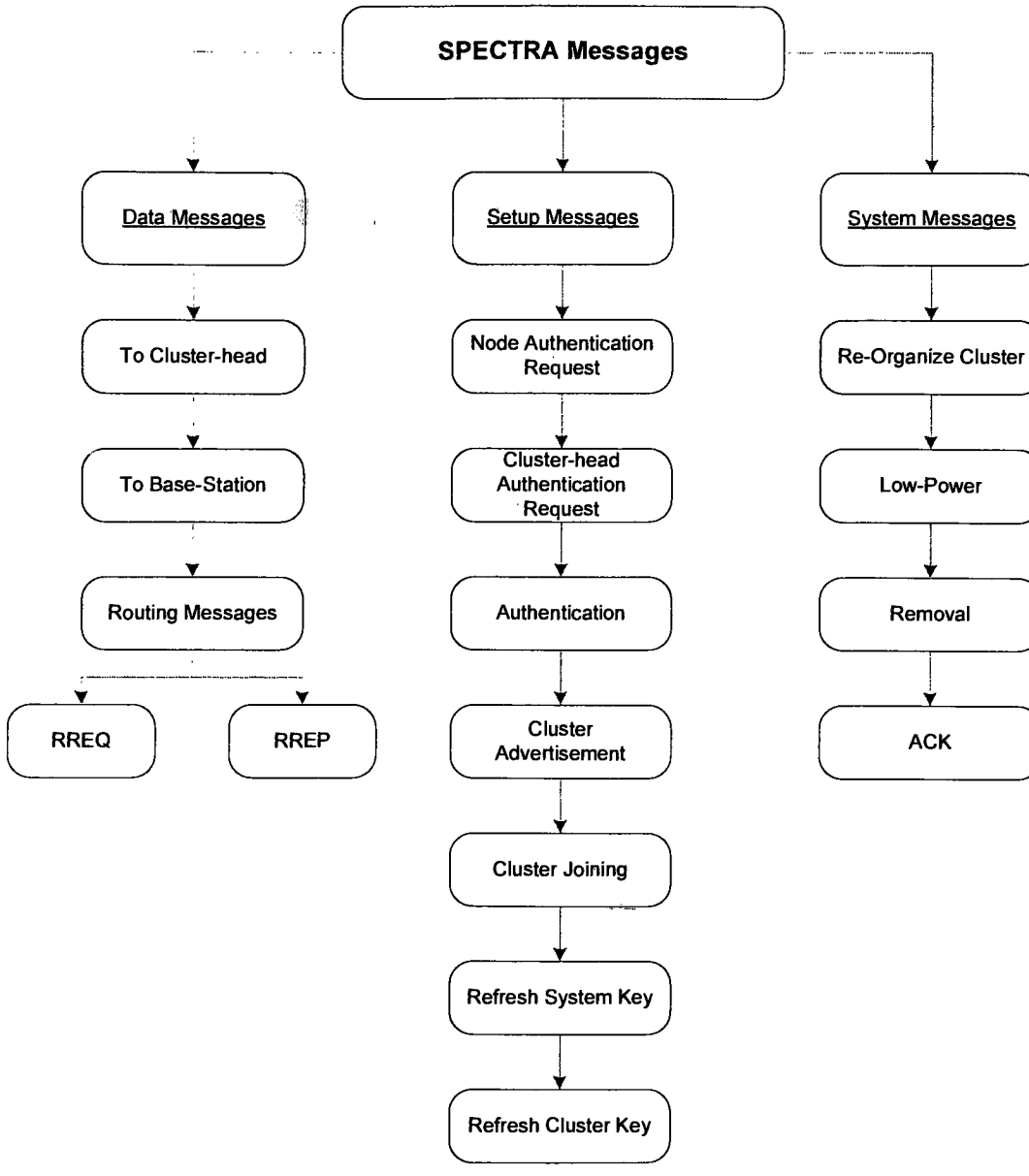

Figure 7: SPECTRA Message Hierarchy

3.8.1 Message Headers

Every message sent passes through the routing and security layer, where it gets a routing header that includes:

- Message Source
- Message Destination
- Number of Hops
- List of Hop IDs

Whenever any node receives a packet, the first thing it does is use the latest *system key* to decrypt the routing header, and figure out if the message is intended for that node, or needs to be forwarded to the next hop on the path to its destination.

This routing header is constructed from the routing table of the node sending the message. If that node does not have the latest *system key*, then any packets that it sends will be rejected and ignored. In order to actually send a real data or command message, the actual route to be taken to the destination must be known. If this route is not known, then a routing request message will be sent, and the initial message will be kept in a waiting list until a route to its destination is known. Whenever an *RREP* packet is received or eavesdropped, the list of waiting messages is scanned to verify if any of them have become deliverable as a result of newly discovered routes.

3.8.2 Setup Messages

These are messages used during the *system setup phase* and for system expansion, when nodes or CHs are added after initial deployment. They are used for the formation of clusters, and for initial and post-deployment authentication. The following sections go over each type of system message in detail.

3.8.2.1 Message — "*Node authentication request*"

Nodes request authentication from the BS. This request contains the node ID. The data portion of the message is encrypted with the node's *personal key* while the routing information is encrypted with the *initial key*. This message registers the node with the BS. In response, the BS replies with an authentication message. Each *personal key* can only be used to request a *system key* once.

3.8.2.2 Message — "*CH authentication request*"

CHs authenticate the same way as nodes, but also identify and register themselves as CHs with the BS. The data portion of this request is encrypted with the *personal key* of the requesting CH, which authenticates the requester with the BS. The routing portion of this request is encrypted with the *initial key*. In response to this request the BS replies with an authentication message. Each *personal key* can only be used to request a *system key* once.

3.8.2.3 Message — "*Authentication*"

This message is sent back by the BS in response to a request for authentication. The data portion of this message is encrypted with the *personal key* of the authenticating node or CH. The routing information is encrypted with the *initial key*. If the authenticating party is a node, then the message contains only the latest *system key*. If the authenticating party is a CH, then this message also contains a *cluster key*.

3.8.2.4 Message — "*Cluster advertisement*"

CHs broadcast this message, encrypted with the *system key*, to all nodes during *initial system setup*. A *cluster advertisement* message contains the CH's ID and the *cluster key*. This message type is also utilized during the post-deployment addition of a CH. Nodes use the RSS (received signal strength) of these advertisements in order to determine which cluster they should join.

CHs listen to each other's advertisements, and use this to construct their neighbor tables of nearby CHs. This dual usage of messages reduces the communication needed for *system setup*. A single set of broadcast advertisements allows for the organization of nodes, the distribution of *cluster keys*, and the construction of neighboring CH neighbor tables.

3.8.2.5 Message — "*Cluster joining*"

In response to CH advertisements, Nodes broadcast a request to join the cluster from which they received the highest signal strength. This message is encrypted with the *system key*, and in response, the CH adds the joining node to its cluster member registry. CHs also register the requesting node as a member of their cluster in response to this message. This registry of nodes in the cluster is eventually forwarded to the BS after the completion of the *cluster organization phase* of *system setup*.

The header of a *cluster joining* message is encrypted with the *system key*, and so these messages are also used by nearby nodes to construct their neighboring node tables to facilitate the creation of multi-hop routes to the CH.

3.8.2.6 Message — "*Refresh system key*"

Periodically, the BS sends out new *system key*s. Each of these periods is defined as an epoch. The new *system key* is always encrypted with the previous *system key*. This means that any node must have the previous *system key* in order to get the latest *system key*. This message is broadcast to the entire network and each individual node authenticates itself by decrypting the new *system key*. This type of message is also never forwarded.

3.8.2.7 Message — "*Refresh cluster key*"

This message is sent by a CH in response to the discovery of a missing or compromised node (see section 6.5). This message can also be sent in a periodic fashion if the network environment requires additional security. The message is encrypted with the *system key* and contains a new *cluster key*.

3.8.3 Data Messages

As mentioned earlier, the purpose of a WSN is to gather data. Nodes generate data from their sensors, encrypt it with their *cluster key*, and forward it to the CH. The CH is responsible for data collection, as well as forwarding the aggregated data to the BS. The actual contents of a data message are the data value and the node ID from which the data originated.

3.8.3.1 Message — "*Node-to-CH*"

This is the most common and frequent type of message that will pass through the system. All nodes periodically generate these messages, and forward them to their CH, encrypted with the *cluster key*. Data messages in multi-hop routes to the CH are eavesdropped by nodes along the way. If this data is found to duplicate the forwarding nodes' data, then the message is forwarded. When this occurs, the timer on the forwarding nodes data generation is reset, so duplicate data will not be sent.

3.8.3.2 Message — "*CH-to-BS*"

CHs collect and aggregate data from the nodes in their cluster, and periodically send all of this data to the BS. This packet contains multiple data messages and the cluster ID. If this message was in response to a data query, then it also contains the query ID number. This data aggregation is encrypted with the *system key* and can be forwarded to the BS along a multi-hop route of CHs.

3.8.3.3 Routing Messages

In order to get a valid route to a given destination, such as the BS, a route must be established. Routes are established on a reactive basis when needed, usually during the *initial system setup phase.*

3.8.3.3.1 Message — "*RREQ*"

When a node needs to communicate with a node for which it does not have an entry in its routing table, that node sends a Route Request message (*RREQ*) to all of its

neighbors. The neighbors then forward the *RREQ* to their neighbors, each appending their node ID. This process continues until the *RREQ* is received by the destination, which replies to the first *RREQ* message received.

3.8.3.3.2 Message — "*RREP*"

Route Reply messages (*RREP*) are generated in response to an *RREQ*, by the destination node, or a node that knows a route to the destination. An *RREP* contains the complete route from source to destination. This message then follows the path that it specifies back to the requesting node (the source).

Unlike *RREQ*s, *RREP* messages are eavesdropped by all nodes that receive them and the route in the *RREP* is extracted in order to fill in the intervening nodes routing table. This eavesdropping takes place to reduce unnecessary routing communication.

3.8.4 System Messages

System messages are those messages that pertain to the system operation of the SPECTRA network. System messages handle aspects of the system such as power management, cluster re-organization after CH death or compromise, and acknowledging received messages.

Cluster re-organization occurs in response to CH death or compromise. A CH notifies the BS once its power has dropped below a certain threshold (this is a *low-power* notification message - 3.8.4.2). After the CH dies, the BS broadcasts a cluster-reorganization message to the corresponding cluster, encrypted with the old *cluster key*.

Like CHs, nodes send *low-power* notification messages. Instead of sending this message to the BS, they send it to their CH. When the node finally dies, the CHs broadcasts a *Removal* message to its cluster, encrypted with the *cluster key*.

The *Removal* message instructs all receiving parties to remove the specified node from their routing tables. The *Removal* message also advises effected nodes to proactively establish new routes, replacing those routes that contained the dead node. In the case that a *Removal* message was unable to be sent, the SPECTRA network removes nodes and CHs automatically if they are not heard from (assumed as compromised, see section 6.6).

An *ACK* is sent in response to every valid successfully-decrypted non-broadcast message. NAKs are not used in SPECTRA in order to avoid replying to flooding by the enemy. If NAKs were used, flooding of incorrectly encrypted or erroneous messages would result in excessive communication leading to reduced system lifetime.

3.8.4.1 Message — "*Re-organize Cluster*"

When a CH dies or is compromised, the BS broadcasts a *re-organize cluster* message (containing a new *cluster key*) to the corresponding cluster, encrypted with the latest *system key*. This message informs the recipients of the CHs death and advises them to re-organize. The cluster re-organizes itself by appointing the node with the largest remaining power as the new CH. Then newly appointed CH will then proceed to integrate itself within the existing backbone through *RREPs* and *RREQs*.

3.8.4.2 Message — "*Low-Power*"

The message is generated by both nodes and CHs once their remaining power drops below a certain threshold. The only difference between a node generated and CH generated power message is the recipient. In the case of a node, the message is sent to the CH, encrypted with the *cluster key*. While in the case of the CH the message is sent to the BS, encrypted with the *system key*. This message informs the recipient of the nodes power level and causes the recipient to set a timer that is three times the predicted remaining lifetime of the node. The timer is set to three times the predicted lifetime because survey work has shown that most nodes die within this timer [45]. In the case that a node or CH die before this message is generated, the *removal* message is in place to handle this situation (3.8.4.3).

3.8.4.3 Message — "*Removal*"

This message is generated by the detecting node or (nodes detect compromises from a failure to respond to a message) when the timer created in response to the *low-power* message expires. If the message is generated at the BS, it is encrypted with the *system key* and broadcast to all CHs (the SPECTRA network backbone). If the message was generated at a CH or node, it is encrypted with the *cluster key* and broadcast to its cluster. This message is responsible for informing all receiving parties to remove the dead node (specified in message) from their routing tables. The *removal* message also advises all recipients to proactively establish new routes, replacing those routes that contained the dead node.

3.8.4.4 Message — "*ACK*"

Every non-broadcast successfully decrypted message must be *Acknowledged* (*ACK*ed) at every step along the way to its destination. This ensures reliable communication and reduces re-sending within the network. If an *ACK* is not received within some period of time, the message must be re-sent. An *ACK* is not sent if the received message is encrypted incorrectly or if it is encrypted with an out-of-date *system key*.

3.9 Summary

This chapter provided an in depth coverage of the SPECTRA architecture and the components within it. The specific functionality associated with clusters, CHs, and nodes was discussed. This chapter also went through in detail the three types of messages: setup, system, and data messages that exist within the SPECTRA protocol. It also provided a brief overview of routing and how routes are established within the network. The information portrayed in this chapter provides a better understanding of the protocol architecture and its specific components. As a result, *system setup* and functions can be more easily understood. The following chapter will go through the initial setup of routing and security within a SPECTRA network.

Chapter 4. Initial System Setup

SPECTRA *system setup* consists of three phases: the *authentication phase*, *cluster organization phase*, and *route establishment*. Together these three phases comprise the *initial system setup phase*. Each of these *system setup* phases builds upon the previous phase and must be completed before the following phase can commence. A detailed description of each phase is described in the following sections.

4.1 *Authentication Phase*

In order for any node or CH to participate in the SPECTRA network, it must be authenticated. A diagram depicting an overview of the a*uthentication phase* can be seen in Figure 8. A node is authenticated by having the latest *system key*. In order to get the latest *system key*, a node broadcasts a request to the BS, encrypted with that node's *personal key* (Figure 9) and the *initial key*. The BS knows that the node is authentic because the node has the *personal key* associated with its node ID. The BS replies to the node by broadcasting back the latest *system key*, encrypted with the *initial key* and the *personal key* of the requesting node (Figure 10). The node receives and decrypts the *system key*, and can then attempt to join a cluster.

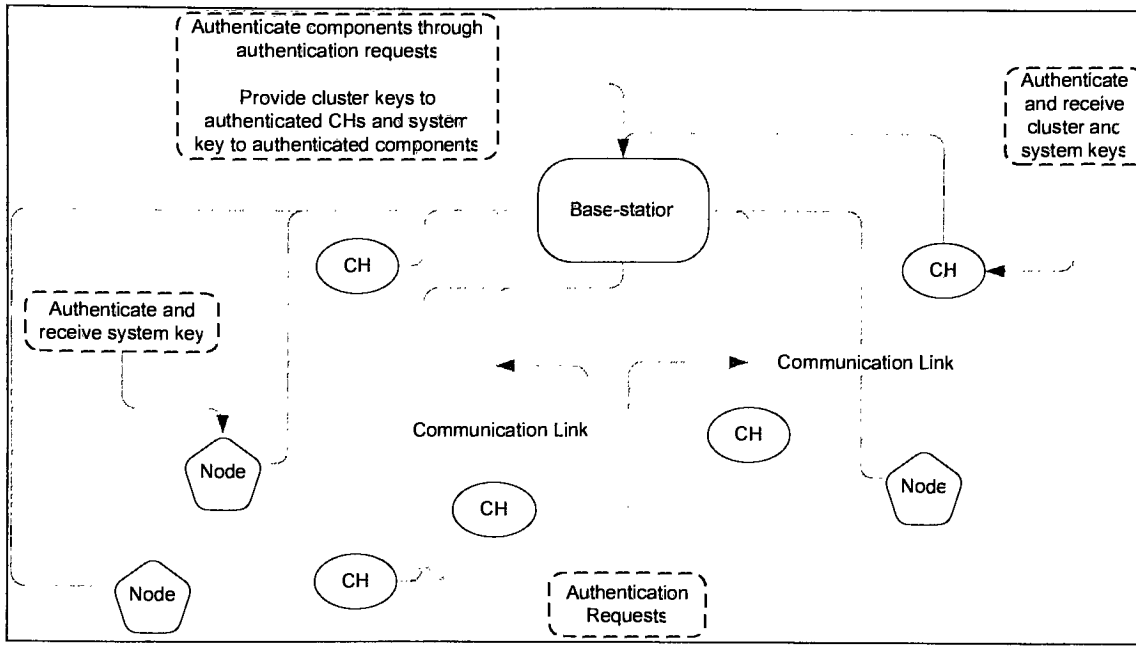

Figure 8: Overview of *Authentication Phase*

When initially requesting the latest *system key*, CHs authenticate themselves in the same fashion as nodes, through their *personal key* and the *initial key*. They also identify themselves as CHs, and request a *cluster key* which they will use to organize a cluster. CHs receive both the latest *system key* and a *cluster key* in reply to their *authentication request.*

It is important to note that only the data portion of the *authentication request* is encrypted with the node's *personal key* while the routing information is encrypted with the *initial key* (see sections 3.8.2.1-3.8.2.3). After this initial authentication, and for the rest of its lifetime, a node will continuously receive and decrypt the latest *system key*. This ensures continuous authentication.

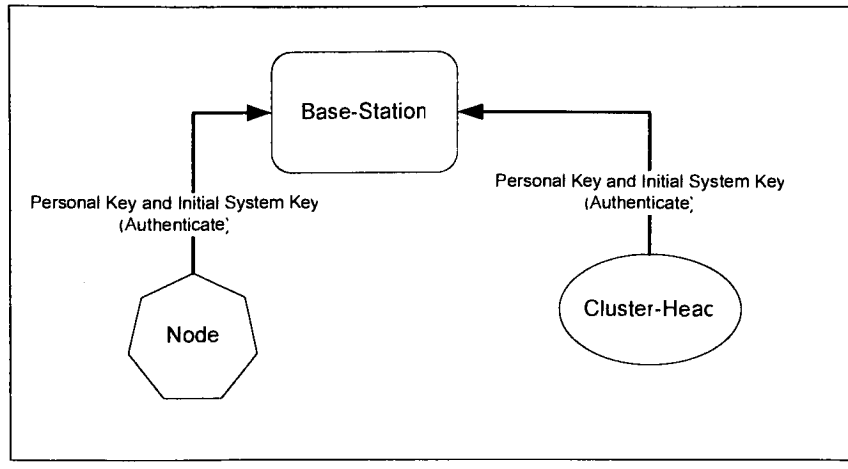

Figure 9: CH and nodes requesting *System key* and undergoing initial Authentication

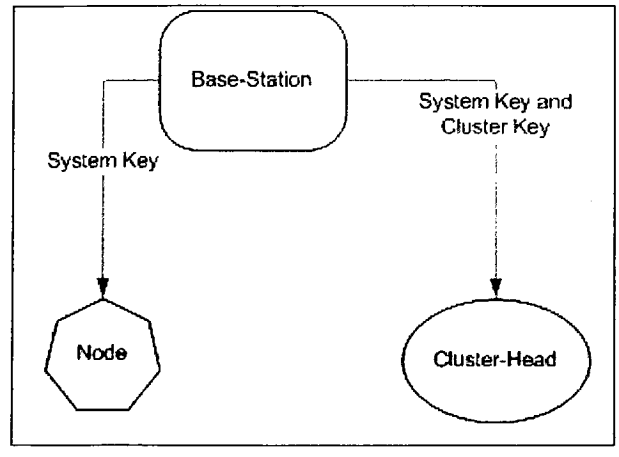

Figure 10: BS sending *system key* to Authenticated nodes, *System key* and *Cluster key* to CHs

4.2 *Cluster Organization Phase*

The *cluster organization phase* of SPECTRA sets up the network topology through the creation of clusters. A diagram depicting an overview of the *cluster organization phase* can be seen in Figure 11. Clusters are formed by using an existing energy efficient clustering algorithm. SPECTRA incorporates the load balancing clustering algorithm, which can form clusters with balanced traffic distribution among clusters [46].

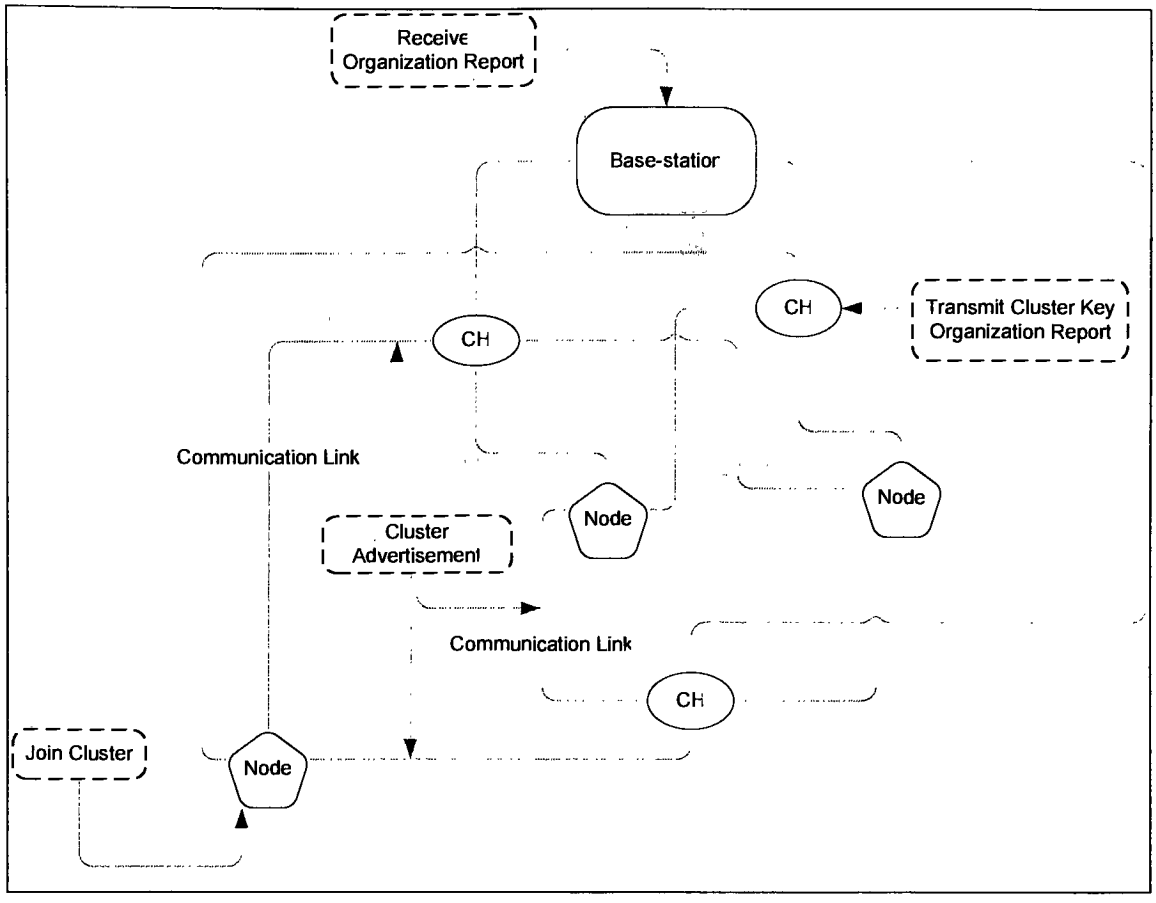

Figure 11: Overview of *Cluster Organization Phase*

Once a CH is authenticated, it broadcasts an advertisement, encrypted with the latest *system key* (Figure 12). Advertisements contain the cluster ID number, and the *cluster key*. Nodes listen to these advertisements and record their RSSs. The strongest recorded RSS is associated with the nearest CH, and the node sends a cluster join message to this CH, encrypted with the *cluster key*. The *cluster key* is received through the *cluster advertisement*. In the case that a node receives a few advertisements after some delay and is already integrated into a cluster, a node can simply change its cluster based on the new advertisements. This is achieved through the same process as post-deployment addition (see Chapter 7).

As *cluster joining* requests are received, the CH adds those nodes to its cluster member registry (Figure 13). The CH keeps a counter that is reset whenever a node joins its cluster. When the counter expires, the CH sends a cluster organization report to the BS, encrypted with the *system key*. The cluster organization report is complete with the cluster ID, the CH ID, the current *cluster key*, and the cluster member registry (Figure 14). The cluster member registry is a list of all nodes within a given cluster.

The BS keeps track of network topology through the cluster member registry of each CH. Whenever there is a change in the topology of a cluster, a new cluster organization report is sent to the BS. This knowledge is used in the event of CH compromise in order to re-organize the cluster.

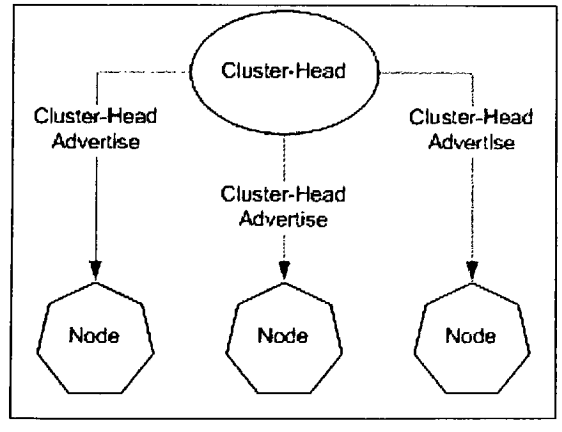

Figure 12: CH broadcast an advertisement to all nodes in the network

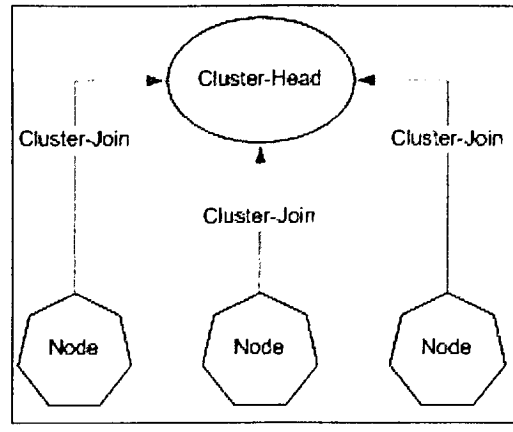

Figure 13: Nearby nodes respond to advertisement with a Cluster-Join message

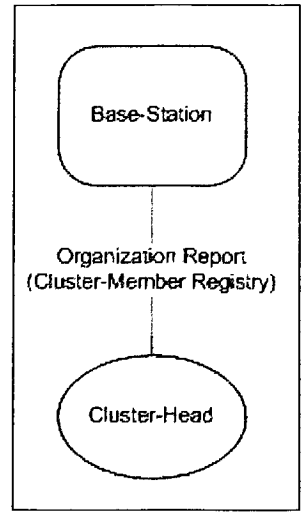

Figure 14: CH updates BS with a Cluster-Organization Report

4.3 *Route Establishment*

This phase of the protocol is responsible for setting up the communication routes for inter-cluster and intra-cluster routing. A diagram depicting an overview of the *route establishment phase* can be seen in

Figure 15. After clusters are organized, but before the CH sends its first cluster organization report, CHs find a route to the BS. If the BS is not one of its neighbors then the CH broadcasts a Route Request (*RREQ*, see section 3.8.3.3.1 and Figure 16) message. A neighbor is defined to be a node who's RSS is above a certain threshold, and every hop

of every multi-hop route must occur between neighbors. This is a dynamic feature that is implemented by the MAC layer, which is not in the scope of this work.

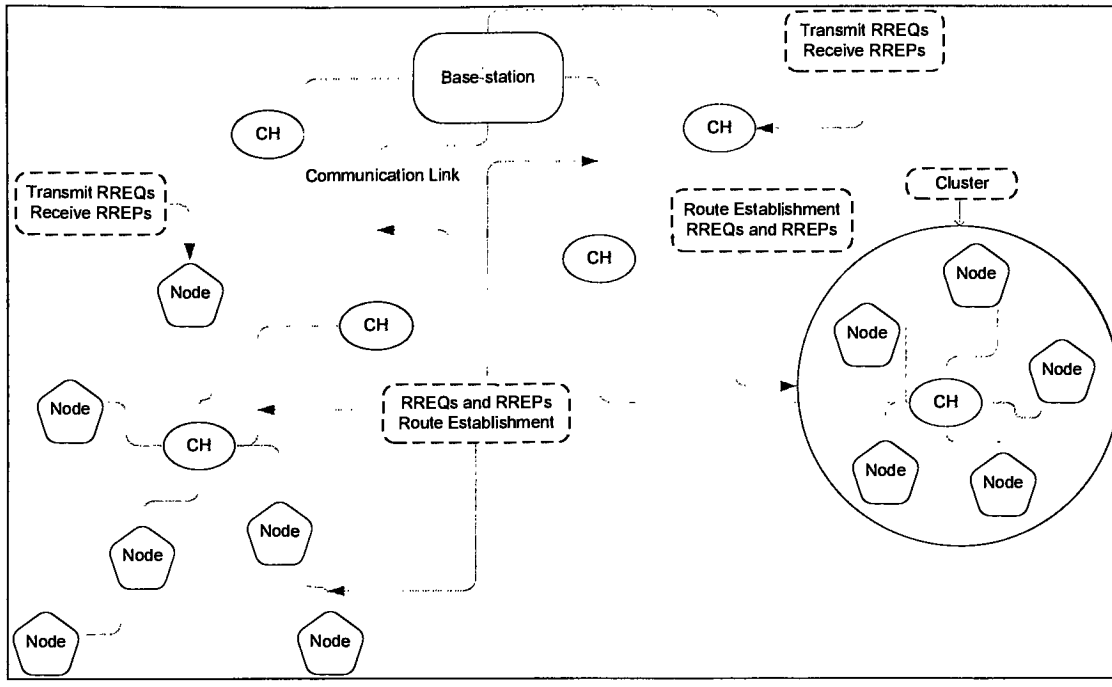

Figure 15: Overview of *Route Establishment Phase*

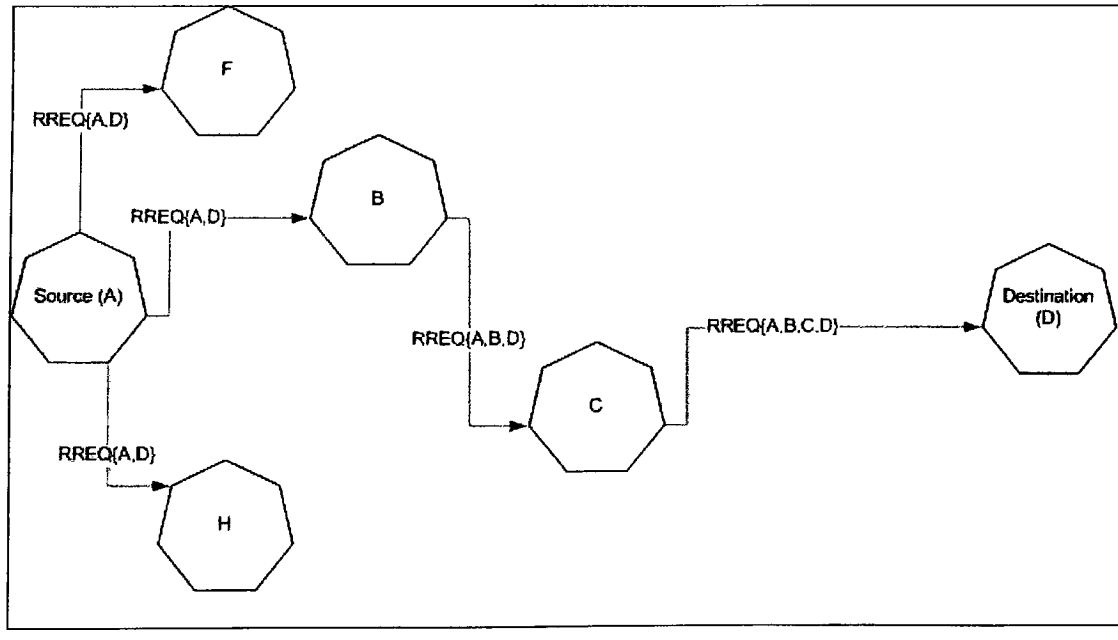

Figure 16: *RREQ* propagating through network to Destination node

All routing messages are broadcast and CHs keep track of the sequence number of each message. When a CH receives a *RREQ* message, it checks to see if the requested destination is one of its neighbors. If the receiving node is the destination of the *RREQ*, then it generates a Route Reply (*RREP*) message containing the whole route from source to destination (Figure 17). Only the first received *RREQ* is replied to. All following *RREQ* messages with the same sequence number are ignored.

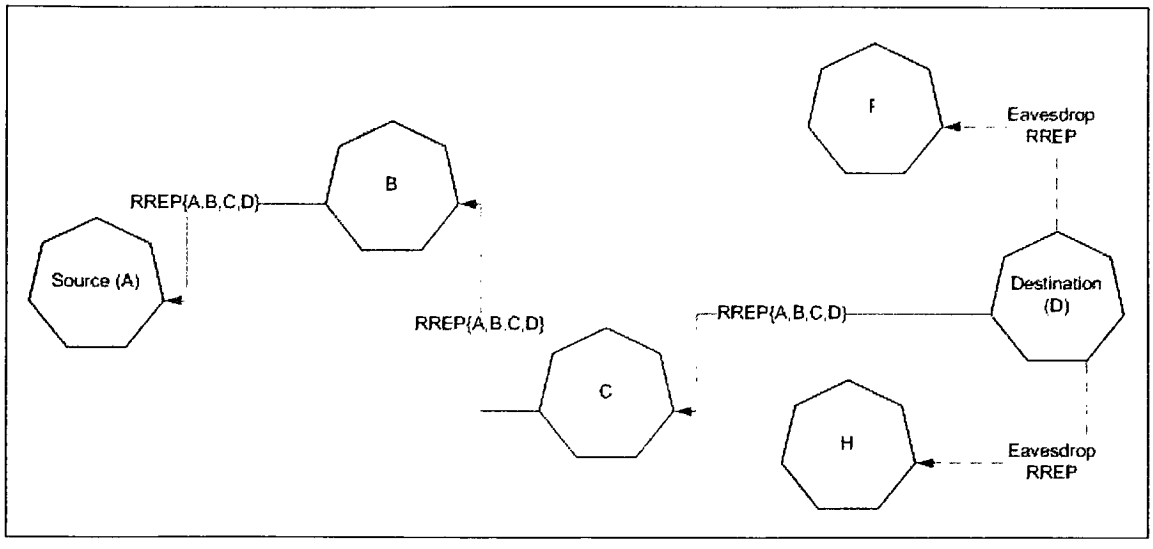

Figure 17: *RREP* generated at Destination and sent back to Source, eavesdrop by surrounding nodes

If the current recipient is not a neighbor of the requested destination, then it forwards the *RREQ* to all of its neighbors through a broadcast encrypted with the *system key*. It appends its own node ID to the route contained within the *RREQ* before forwarding the message. In the event that the *RREQ* is intended for one of the neighbors of the current recipient, the modified *RREQ* is forwarded only to the destination.

This route organization process is used for both CH to BS routing, and for node to CH routing within a cluster. Like *RREQ* messages, *RREP* messages are encrypted with the *system key*, allowing both nodes and CHs to eavesdrop on routing information (Figure

17). This allows them to fill in their own routing tables without sending additional *RREQ* messages.

4.4 Summary

This chapter stepped through the three phases of *system setup*: the *authentication phase*, *cluster organization phase*, and *route establishment*. The *authentication phase* is responsible for authenticating all nodes and CHs. The *cluster organization phase* is responsible for organizing clusters within a SPECTRA network. Finally, the *route establishment phase* is responsible for creating communication links between the nodes, CHs, and BS of a network. Knowledge of these phases provides an understanding of how the system is initially setup. With *initial system setup* understood, the next chapter steps through the operations of a SPECTRA network after it has been setup.

Chapter 5. System Operation after *Initial system setup*

As discussed in Chapter 4, the *initial system setup* consists of authenticating network members, forming them into clusters, and establishing communication links between them. The *system operation phase* of the SPECTRA network commences after the *system setup phase* has been completed. The normal activities of a SPECTRA network are primarily concerned with data movement and continuous authentication. These activities include: data collection, data aggregation, data transmission, continuous authentication, and eavesdropping. This chapter goes through the functions of the network after *initial system setup.*

5.1 Data Collection, Aggregation, and Transmission

All data is generated at the sensor node, in response to an event or time period. When a node has new data it forwards that data to the CH periodically or in response to a data request message. The data follows a direct or multi-hop route to the CH, where it is stored and aggregated (see Figure 18).

The CH maintains a collection of the most recent data generated by each of the member nodes in its cluster. This data is aggregated in order to avoid duplication and is later forwarded to the BS periodically or in response to a data query. Data duplication is avoided by truncating sample values that are similar and come from the same region of the sensor network. The specific data aggregation policy, which includes data duplication strategies, is implemented at the application layer of the CH. SPECTRA does not contain any native data aggregation policy, but supports any reasonable data aggregation policy. Third party data aggregation algorithms, such as the CAG algorithm [52] can be easily

integrated with the SPECTRA protocol. This can be done because data aggregation is not natively a part of the security or routing layer of the protocol stack. Therefore, because data aggregation can be implemented in a different layer, it can easily be integrated with SPECTRA [45].

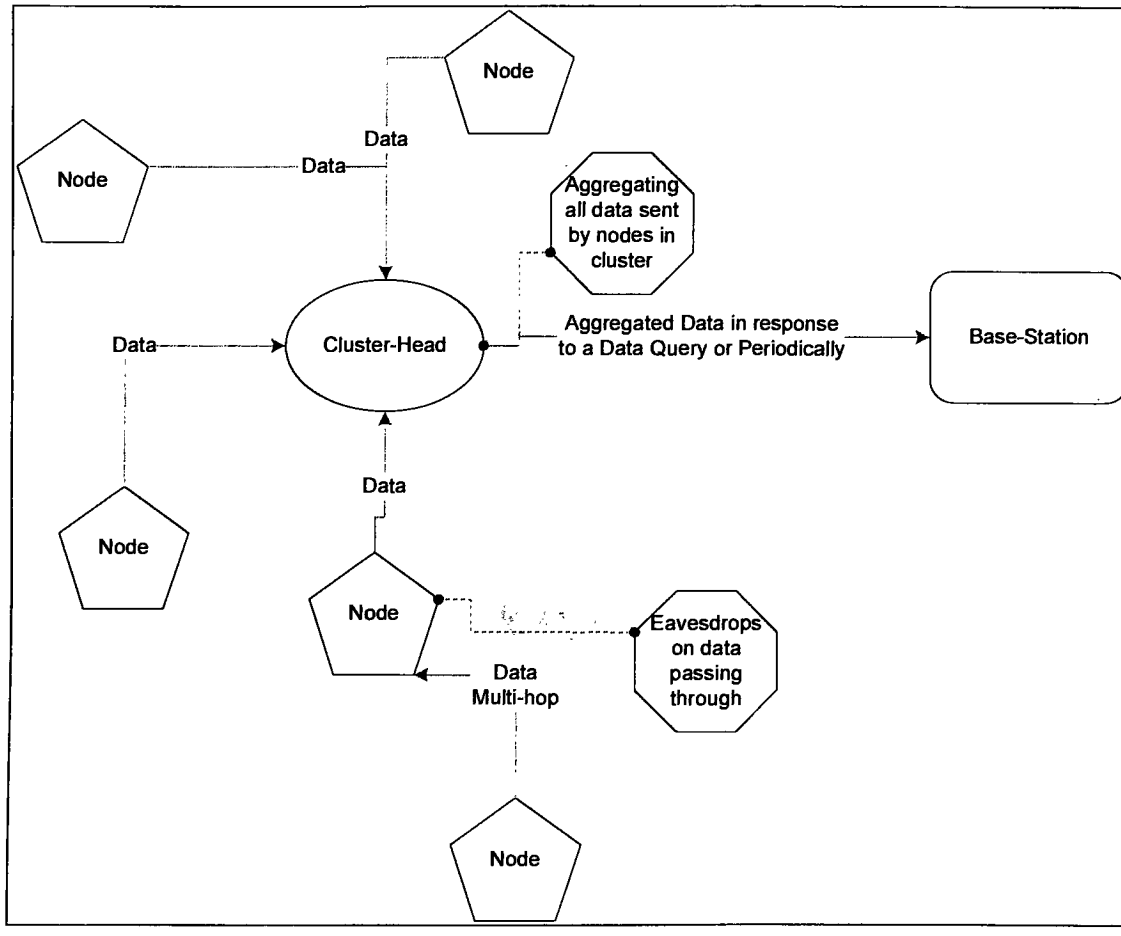

Figure 18: Diagram depicting single-hop and multi-hop data aggregation

When aggregated data is sent to the BS, it is encrypted only with the *system key*. This allows other CHs along the CH backbone of the SPECTRA network to eavesdrop on the current transmission in order to support further data aggregation.

5.2 Continuous Authentication

All nodes and CHs in a SPECTRA network are continuously and periodically authenticated. This authentication is achieved through the periodic refreshing of the *system key*. A *system key* is only valid for a period of time that is referred to as an epoch. At the start of every epoch the latest *system key* is broadcast three times in rapid succession by the BS, encrypted with the previous *system key* (Equation 8 shows the expression for a refreshed *system key*). The *system key* is broadcast three times in order to reduce the effects of wireless errors and the resulting chances of a node failing to authenticate.

Any message whose routing header is not encrypted with the latest *system key* is disregarded and not *ACK*ed. Therefore, a node will be totally ignored by the rest of the SPECTRA network if the node does not have the latest *system key*.

It is assumed that node compromise will take longer than an epoch. This guarantees that a node tampered with by the enemy does not have the latest *system key* when it attempts to rejoin the network. The node cannot attempt to rejoin the SPECTRA network, because each *personal key* can only be used once to acquire the latest *system key* (this is generally done during *system setup*).

$$\underset{one-way}{\overset{f}{\rightarrow}}\left(PRNG\left(K_{current\ K_{System}}\right)\right)=K_{refreshed\ System}$$

Equation 8: Expression for refreshed *system key* (see Table 1 for symbol definition)

5.3 Eavesdropping to reduce redundancy

Eavesdropping occurs in several parts of a SPECTRA network, reducing unnecessary communication. Eavesdropping occurs on data messages on their way to the CH, on data messages on their way to the BS, and on all *RREP* routing messages.

Data messages within a cluster are eavesdropped by any nodes that they pass through on their way to the CH. All data messages within a cluster are encrypted with the *cluster key*. Nodes decrypt and examine the contents of all data messages that pass through them. This allows nodes to avoid sending data messages regarding or duplicating events that other nodes in the cluster have already reported.

Data aggregation messages sent by a CH to the BS along the CH backbone are also eavesdropped by all intervening CHs. This allows for further data aggregation and reduces data duplication and communication.

Like all routing messages, *RREP* routing messages are encrypted with the *system key*, allowing them to be eavesdropped. This allows every node that hears them to decrypt the *RREP* message, and append the corresponding route contained to their routing tables.

5.4 Summary

To summarize, a SPECTRA network is primarily responsible for collecting and transmitting data in a secure fashion once *initial system setup* has been completed. Eavesdropping is conducted within clusters in order to reduce the amount of data being transmitted. This data is transmitted securely by encrypting the data messages with keys. The data being received by the BS can be trusted because the entire network is authenticated periodically through the *system key*.

This chapter along with Chapters 3 and 4 provided a complete understanding of how the SPECTRA network is setup and works along with the functions of every component. This knowledge allows for the security features of SPECTRA to be better understood and how they fit into different aspects of the network. Chapter 6 is devoted to security and how it is achieved through keys and authentication.

Chapter 6. System Security

Security is the focus of the SPECTRA network, and the algorithms raison d'etre. Security is achieved entirely through the use of symmetric key cryptography. SPECTRA is innovative in its use of multiple keys for encrypting each message. This makes node compromise and key compromise intricate for the enemy. Not only must the right keys be known, but it must also be known in which order to apply them to a given message.

The focus of this chapter is the security aspect of SPECTRA. This chapter goes through encryption, confidentiality, dual keys, and authentication. Global authentication is an extremely important aspect to the SPECTRA protocol because it confirms the validity of network members periodically. This feature was loosely inspired by the SPINS protocol as discussed in Chapter 2. This chapter also focuses on the system response to compromised nodes and CHs. This chapter starts out by discussing the foundation of any security protocol: keys and encryption.

6.1 Keys and Encryption

The SPECTRA network utilizes multiple keys to achieve security, authentication, and confidentiality. Due to the limitations of sensor nodes, all keys within SPECTRA are symmetric. Symmetric keys provide security, but are simpler, smaller, and computationally less intensive then asymmetric keys.

SPECTRA uses three main keys: the *system key*, the *cluster key*, and a *personal key*. The *system key* is used for global authentication purposes and is periodically refreshed. The *personal key* is used for initial node authentication during the *system setup*

phase. The *cluster key* is used for security within a cluster, and is used to encrypt the data portions of all messages exchanged within the cluster.

Encryption in SPECTRA is achieved natively through one-way radix hash functions. One-way hash functions are utilized because of their computational ease, low memory and resource overhead, resulting in an infeasible reverse engineering algorithm to determine the original data that was applied to the function. They provide good encryption with little strain on resources; therefore, one-way hash functions are extremely attractive for use in WSN security [30]. SPECTRA is not limited to a single encryption algorithm and was designed to support various methods. The choice of encryption is dependent on the application and the network environment (inhospitable environments call for superior encryption algorithms). However, SPECTRA was designed with one-way hash functions in order to prolong system lifetime. It is important to understand how hashing works in order to fully comprehend encryption in SPECTRA.

Hashing is a transformation that represents a string of characters or numbers with a shorter fixed length value. The hashing algorithm is called a hash function, the resulting hash value derived by the hash function can be thought of as a "mixed up" version of the original value [5]. The digital signature or key is transformed by the hash function, resulting in a hash value. This new value is now used as the encryption key. The resulting hash value is used with the hash function to index the original value or key in database applications. However, in the case of encryption, the resulting hash value is the derived key that will be used for encryption. Therefore, hashing is always a one-way operation [5]. Good hash functions produce unique hash values for each input. Therefore, the same hash value will not be generated by two different inputs (characteristic of hash functions

[5]). Hash functions also hold the property of multiple hashes for a single value. This is done by breaking up the original value that is applied to the hash function into blocks. These blocks are then transformed individually using the hash function and concatenated. The concatenation is then hashed again. Each block is the same size, and if remainders occur they are included in a padded block or discarded. This results in a far more secure transformation, making reverse engineering even more difficult. The diagram below shows the concept of blocked hashing.

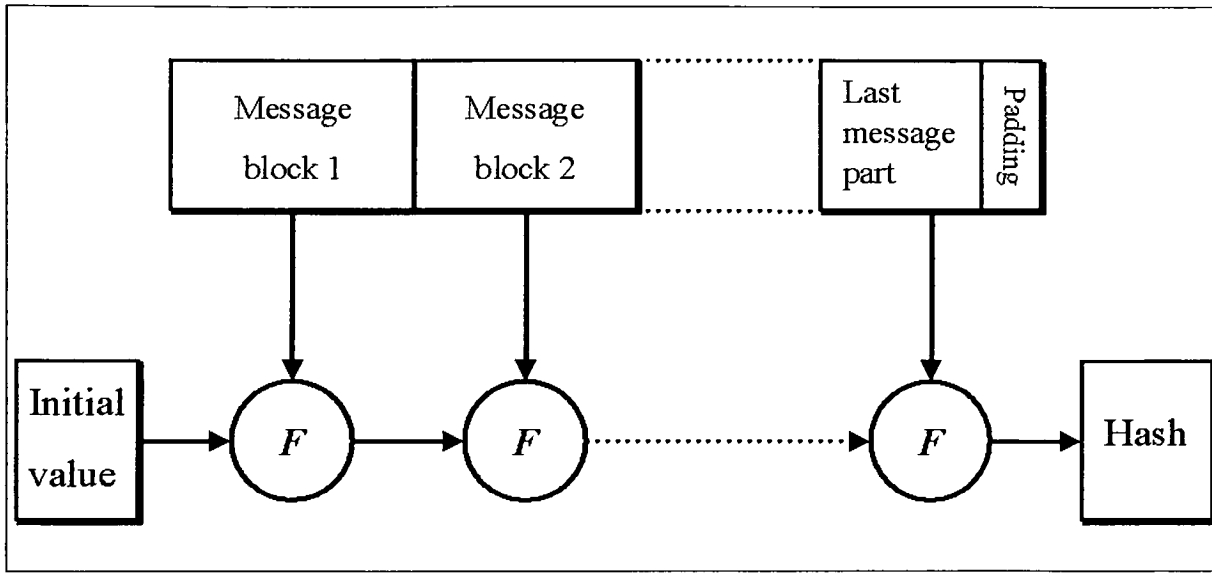

Figure 19: Multi-blocked hashing [5]

SPECTRA implements a one-way radix transformation hash function. This particular hash function takes in a digital value that allows it to change base (or radix). Therefore, a decimal or octal key can be transformed into a hexadecimal key (hash value). This creates an exceptionally infeasible reverse engineering algorithm because the original base and the final base are different [5].

6.2 Authentication

During the *initial system setup phase*, SPECTRA achieves authentication through each node using its *personal key* and *initial key*. Once the *initial system setup phase* has completed, SPECTRA authenticates the entire system by periodically refreshing the *system key*.

All nodes and CHs are frequently and periodically authenticated. This global authentication is achieved by periodically refreshing the *system key*. A node's or CH's *personal key* and *initial key* must be used in order to get the *system key* for the first time. After this key is acquired, the *system key* is used to get all of the later *system key*s, thus achieving periodic authentication.

The concept of authenticating the entire network periodically is based on the SPINS protocol [7]. The SPINS protocol shows the benefits in security through periodic authentication [7]. This feature allows every entity in the network to be confirmed and reduces the chances of compromise because compromise has to occur before the system is authenticated again.

6.2.1 Global Authentication

Global authentication is achieved periodically through the use of the *system key* (the *initial key* is used during initial authentication). The *system key* is broadcast three times in rapid succession to the SPECTRA network at the beginning of every epoch. The latest *system key* is encrypted with the previous *system key*. The *system key* is periodically refreshed and encrypts every routing header. Therefore, a node has to have knowledge of it in order to function within the SPECTRA network. The periodic refreshing

authenticates each node to the BS as well as authenticating it to anyone communicating with it. If a node can decrypt the routing header, it is authenticated. Since the *system key* is dispersed to the whole network and refreshed every epoch, global authentication is achieved periodically. A diagram depicting the broadcast of the latest system key at the beginning of an epoch can be seen in Figure 20.

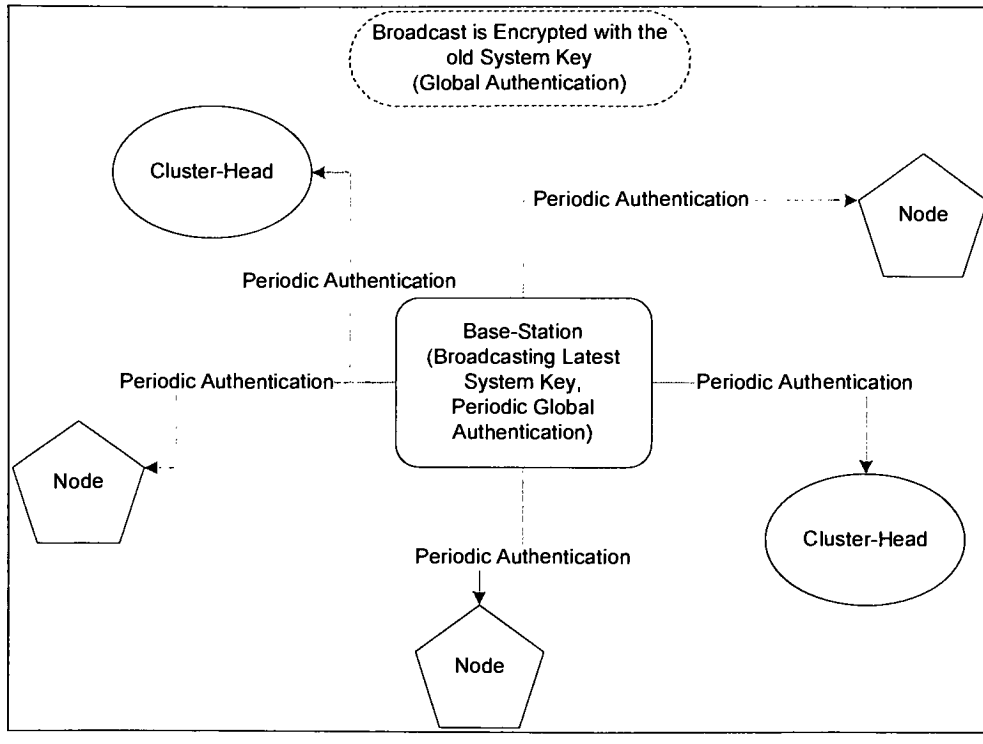

Figure 20: Diagram depicting one of three broadcasts done during Global Authentication

An epoch is defined to be a period of time that is less than the predicted time required for node compromise. The epoch time is dependent on the network environment and is determined by an environment analysis. An epoch is used for scheduling the broadcast of the latest *system key*. At the state of every epoch, the *system key* is broadcast three times in rapid succession. This is done to combat wireless interference, fading, and packet error rates [41]. The assumption that an epoch is less than the time period of

compromise is the foundation of SPECTRA's global authentication. Figure 21 shows how authentication is achieved using an epoch.

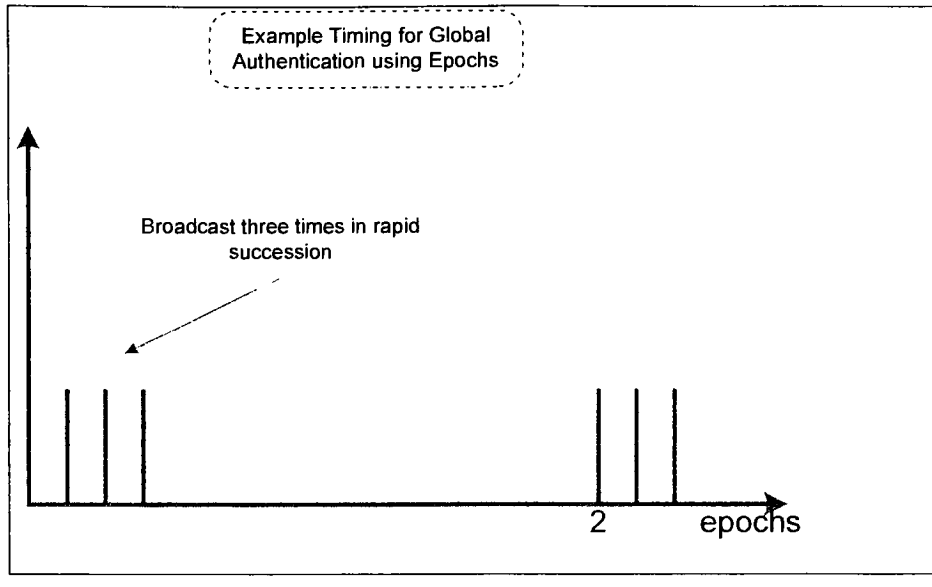

Figure 21: Epochs for Global Authentication

6.3 Dual Key Usage

Dual keys are extremely important in the SPECTRA network because they make compromise exceptionally difficult and provide two levels of authentication. Not only must a compromised node have knowledge of three different keys, but also know exactly when to use them. Also, because of different keys and message sizes, it is extremely difficult to decipher the different portions of the message.

SPECTRA uses two keys to provide confidentiality and authentication at every step in the network. All routing information of any message passed within the SPECTRA network is encrypted with the *system key* (or the *initial key* during the *initial system setup phase* see section 4.1) while the data portion is encrypted by either the *system key*, the *cluster key*, or the *personal key* of the node. Therefore, in order to even function within

the SPECTRA network, knowledge of the current *system key* must be known. If a node is lacking the *system key*, no information can be sent or received (in other words, routing cannot be understood). This provides first level authentication of the node.

The data portion of all messages within the SPECTRA network can be encrypted with different kinds of keys. Thus, a node needs to have knowledge of network topology and understand network functionality in order to use the correct key for decrypting the data portion. This provides a second level of authentication for the node and verifies that the node understands how the SPECTRA network works. If the node lacks knowledge of system functionality, it will be unable to identify which key to use for data decryption.

6.4 Confidentiality

SPECTRA achieves confidentiality through the use of keys and encryption. Every message sent within the SPECTRA network is encrypted. In particular types of messages two different keys are used. As a result, nodes must have knowledge of where to use specific keys and when only one key is needed. This provides confidentiality, authentication, and verifies node identity.

In situations when SPECTRA uses two different keys to encrypt a message, the source node needs to have knowledge of both types of keys and the order to use them. This ensures confidentiality and node identity. The receiving node also requires access to both keys and knowledge on how to decrypt the message. If the message is read successfully, the receiving node transmits an *ACK* message, informing the source of the successful communication.

6.5 Node Compromise

The SPECTRA network can have two types of node compromise. In one situation, the node misses the latest *system key* due to wireless errors and can no longer function in the system since its routing header cannot be decrypted. This node is ignored and eventually removed by the system and will be unable to reenter the SPECTRA network. In the second situation, nodes miss the *system key* due to an enemy trying to infiltrate the network by physically compromising the node. This node is unable to renter the SPECTRA network because it missed the latest *system* key broadcast. The value of global broadcast can be seen in this situation. This is done in order to ensure security. SPECTRA does not distinguish between compromised nodes and nodes that have dropped out due to wireless errors, therefore both situations are handled in the same way.

6.5.1 Node failure due to wireless errors

Nodes are assumed to be compromised when they fail to respond to a message that has been resent to them 5 times (fail to *ACK* 5 times). The node that detects this broadcasts a *removal* message, encrypted with the *system key*, which notifies the SPECTRA network of the compromise. The *removal* message results in the removal of the compromised node from all routing tables of the system. This includes all routes that contain the compromised node. For more detail regarding the *removal* message see section 3.8.4.3.

6.6 Clusterhead Compromise

Like nodes, CHs can be compromised or drop out due to wireless errors. Both situations are treated in the same manner and the CH is removed from the system. A compromised or dropped out CH causes problems for the cluster that it was controlling. In response to this problem, the BS tells the nodes of that cluster to re-organize and appoint a new CH.

6.6.1 Clusterhead failure due to wireless errors

CHs are assumed to be compromised when they fail to respond to a message that has been resent 5 times to them (fail to *ACK* 5 times). The detecting node or CH generates a *removal* message, encrypted with the *system key*, which is broadcast to the SPECTRA network. The *removal* message informs all receiving parties of the compromise. This message results in the removal of the compromised CH from all routing tables and routes in the system. It is important to note that it is far harder to spoof a *removal* message then inject noise into the system. In order to spoof the message, encryption keys must be broken and used before the system refreshes. Noise saturation of the network is something that is handled in the physical layer of the network stack and outside the scope of this work. For more detail regarding the *removal* message see section 3.8.4.3.

6.6.2 Cluster Re-Organization

When a CH is compromised and detected, a *removal* message is broadcast to the system. The BS generates a Re-Organization message in response to this *removal*

message, and broadcasts it to the corresponding cluster (see Figure 22). This message informs the cluster to re-organize itself and appoint the node with the largest remaining power as the new CH. The newly appointed CH then broadcasts itself to the cluster backbone via a *Cluster advertisement* message (see Figure 23).

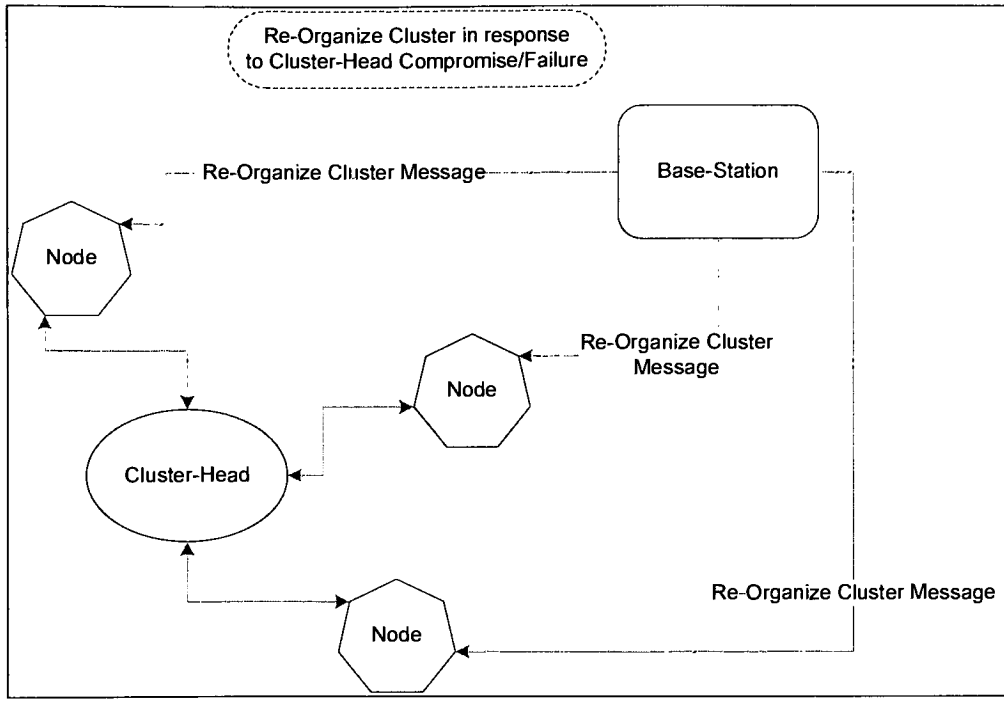

Figure 22: Diagram depicting *Re-organize cluster* Message in response to a *Removal* Message

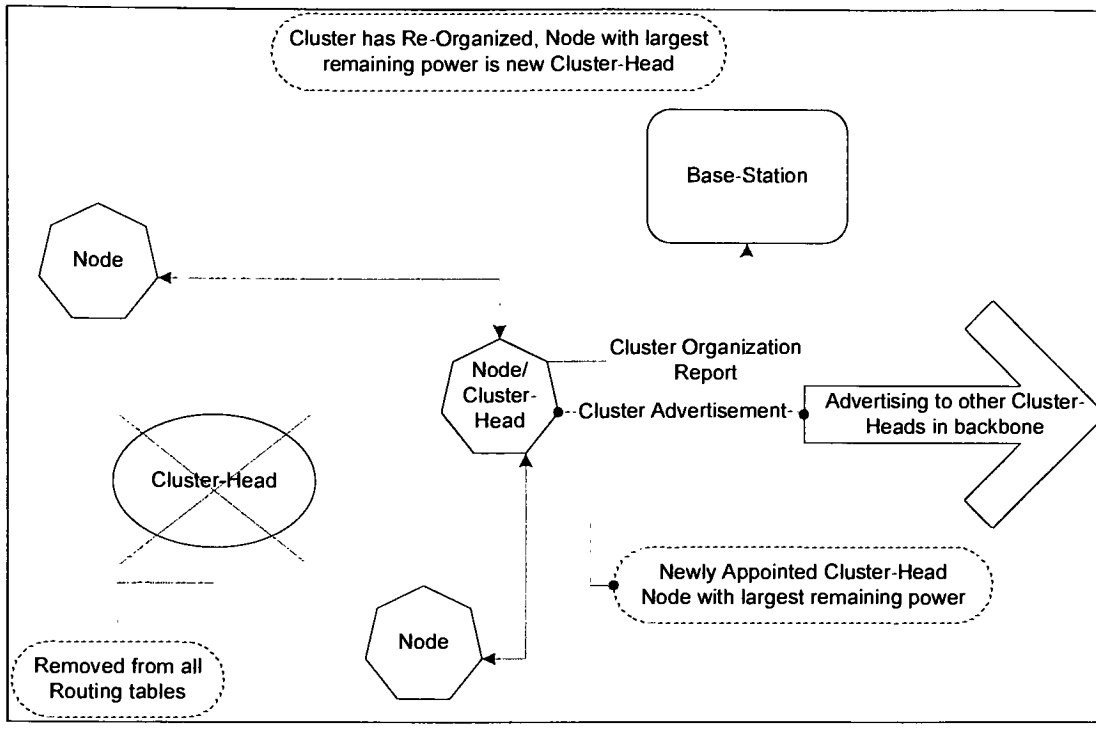

Figure 23: Nodes have Re-Organized themselves and new node has sent a *Cluster advertisement* and a Cluster-Organization Report

6.7 Summary

Given a WSN, it is vital to have an adaptive security protocol, as mentioned in Chapter 2. This chapter focused on how security is provided within the SPECTRA network during *initial system setup* and post *system setup*. This only covers security within a static network. However, SPECTRA is an adaptive security protocol and responds to the dynamic topology natural to a WSN. The following chapter focuses on how SPECTRA is adaptive and its responses to topology changes.

Chapter 7. Adaptive Security to dynamic WSN topology

As mentioned in Chapter 2, dynamic topology is an inherent feature of WSNs due to their resource limitations, in particular power limitations. Therefore, it is imperative that a protocol be scalable and expandable in response to the dynamic topology of WSNs. SPECTRA was designed to be robust in network scalability and expansion. The limited battery lifetimes of nodes and CHs ensure the eventual death of every node in the system. As nodes fail from lack of power, system performance degrades. If no action is taken, the system will eventually cease to function.

To ensure continued system functionality, new nodes and CHs must be deployed as the system ages to replace the nodes and CHs that have died. Node death and the deployment of additional nodes result in a continuously changing network topology. These changes in network topology complicate the routing of messages within the network.

The most important challenge to a security protocol is ensuring that the security of the network is not compromised by node failure and addition. Security is guaranteed by requiring that all nodes joining the network after the *initial system setup phase* authenticate themselves with the BS.

This chapter focuses on how nodes and CHs are added to an existing SPECTRA network in a secure fashion (expansion of network after initial deployment).

7.1 Post-Deployment Authentication

Nodes joining the SPECTRA network after the *initial system setup phase* has completed, need to authenticate themselves to acquire the latest *system key*. Nodes trying

to join the SPECTRA network are deployed with a *personal key* that is used to authenticate the node to the BS (each *personal key* may be used only once to request the *system key*). Authentication is achieved when the node sending a message to the BS requests to join the existing network, encrypted with the node's *personal key* and *initial system key*. If the *personal key* is valid and has not been used for authentication before, the BS replies with the latest *system key*, encrypted with the requesting nodes *personal key* and *initial key*. Once the node has acquired the latest *system key*, it attempts to find and join the nearest cluster by broadcasting a *cluster joining* message (see section 3.8.2.5), encrypted with the *system key*.

Like nodes, CHs trying to join an existing SPECTRA network need to be authenticated. CHs are authenticated in the same way as nodes by sending a request to join message to the BS, encrypted with its *personal key*. If the CH is successfully authenticated, the BS replies with the latest *system key*, encrypted with the CHs *personal key* and *initial key*. Once the CH has been authenticated it sends a message to the BS requesting a new *cluster key*, encrypted with the *system key*. With the *cluster key* and the latest *system key*, the CH now proceeds to organize a cluster (see section 4.2). A more detailed overview of node addition and CH addition can be found in sections 7.2 and 7.3.

7.2 Node-Addition

Nodes trying to add themselves to an existing SPECTRA network need to undergo authentication. Once authenticated, the node broadcasts a *cluster joining* message, encrypted with the *system key*. CHs reply to the request with their cluster ID and *cluster key*, encrypted with the *system key*. The node keeps track of the RSS of all

replies and joins the cluster with the highest RSS (closest cluster). All information regarding other clusters is removed, and only the *cluster key* from the cluster that is joined is kept. The node ID of the new node is then added to the cluster member registry of the CH which is eventually forwarded to the BS (in the cluster organization report).

7.3 Clusterhead-Addition

Once a joining CH has been authenticated by the BS and acquired both a *system key* and a new *cluster key*, it proceeds to set up a cluster. The CH broadcasts a Cluster-Advertisement message that is encrypted with the *system key*. Any node that receives a stronger signal from the new CH compared to the signal from their current CH replies with a *cluster joining* message, encrypted with the *system key*. This message is broadcast and contains the node ID and the CH ID. This message is broadcast, and therefore all CHs will receive it. This enables the node's current CH to remove the node from its cluster member registry, and the new CH adds that node to its registry. Eventually the BS is notified of this change through a cluster organization report.

7.4 Node Death

When a node's available power drops below a certain threshold, that node sends a Node Death message to its CH. The CH then sets a timer to three times the predicted remaining lifetime of the node. The timer is set to three times the predicted lifetime because survey work has shown that most nodes die within this timer [45]. Once this timer expires the CH removes this node from its cluster member registry and broadcasts a notification to its cluster. This message instructs all nodes in the cluster to eliminate the

dying node from their routing tables. Node death handling is completed when the dead node has been removed from all routing tables and the cluster member registry. Nodes may sometimes be removed prematurely because they happen to last longer then the expected timer. However, SPECTRA eliminates nodes in order to maintain security.

7.5 Clusterhead Death

CH death is handled in a similar fashion to node death. When a CH reaches a certain power level, it sends a message to the BS notifying it of its state. The BS sets a timer for three times the predicted remaining lifetime of the CH. When the timer expires, the BS uses a broadcast to notify the corresponding cluster that its CH has died. This causes the nodes in the cluster to re-organize themselves and appoint the node with the largest remaining power to be the new CH. For more information on cluster re-organization, see section 6.6.2.

7.6 Summary

The focus of this chapter was on adaptive security in response to a dynamic topology. Coupled with Chapter 6, a complete understanding of security within SPECTRA is now achieved. The earlier chapters: Chapters 3, 4, and 5 provided a thorough understanding of network components, setup, and functionality. Together, all of these chapters provide a picture of SPECTRA and the inner workings of it. Now that SPECTRA is understood in its entirety, the next step is to simulate the protocol and better understand the benefits along with the advantages over other protocols. The practicality of a protocol cannot be determined until an analysis has been conducted. In the case of

SPECTRA, a simulation analysis was conducted in order to determine its validity and benefits. Chapter 8 focuses on how SPECTRA was simulated and the tools that were used.

Chapter 8. Simulator Overview

SPECTRA was implemented through a Java-based simulation engine. The simulation engine is comprised of JiST [2] (Java in Simulation Time), and SWANS [3] (Scalable Wireless Ad-hoc Networks Simulator). These particular simulation tools were used due to the lack of available simulation tools for WSNs. Out of the prominently existing simulators, these tools provide a more accurate way of tracking time and a more modular interface. Natively the tools are designed for Ad-hoc sensor networks, therefore requiring pervasive modification to be suitable for WSNs.

It is important to understand the individual tools: JiST and SWANS that are used for the simulation along with the simulation design and setup. This gives a better understanding of the simulation results and makes it easier for analysis. JiST was created to simulate time in Java, while SWANS was created to simulate scalable Ad Hoc networks. The rest of this chapter is devoted to understanding the tools that were used in creating the simulation and the implementation itself.

8.1 JiST

JiST was used to create the concept of time within the simulation as well as the tracking of objects (entities) and their current state. JiST is a Java based simulation framework. To implement a simulation in Java through JiST, the programmer must break down the simulation model into the primary simulation objects. Once all the primary simulation objects are determined and broken up, each of these are implemented in a Java class or an entity as JiST calls it. Each class or entity can keep track of its own state, enabling the simulation model to track the state of the object (represented by a class or

entity) within the simulation. Post compilation, JiST rewrites the byte code associated with each class that represents an object in the simulation model [2]. In doing this, JiST replaces recursive method invocations with code that schedules the method to be called during a certain period of time within the simulation, resulting in an event [2]. This particular feature allows for the creation of infinite recursive functions. However, exceptions do not occur since an infinite recursive call is no longer functioning in its native nature but rather as a continuous event. As a result, JiST allows the programmer to create continuous events without crashing the system or the simulation.

8.1.1 Simulation Time in JiST

The idea of time is implemented in JiST by creating what is known as *Simulation Time*. The concept of simulation time was created to control the progress of the clock while the simulation is running. This allows for a more accurate timing analysis compared to current simulators. Current simulators may provide adequate models of network topology, but they depend on the system clock to determine the efficiency of the network topology, MAC protocol, routing protocol, etc [2]. Because current simulators are dependent on the system clock, an accurate estimation of time can only be achieved if the simulation can model time based on the runtime. However, attempting to simulate real time by modifying code to run on a virtual machine is practically impossible due to external optimization techniques that are utilized by various virtual machines. Java imposes more problems due to its implementation of garbage collection (recovers unused memory) tactics that are sporadic and occasional. This process adds unexpected

processing time to the simulation runtime and cannot be compensated for because of its intermittency and variability in amount of time.

JiST implements what is known as "simulation time" which allows the written code to influence or stop the clock. Simulation time allows the programmer to directly control the amount of "time" required by any given process for the purpose of efficiency calculations. This also allows the programmer to schedule the order and the moment in which an event will take place. Time can therefore be modified according to different microprocessors because the ability to control the amount of time required for a given process is now possible. This allows the programmer to benchmark the simulation over various processors if their timing information is available. This is extremely important for a WSN because processor efficiency is extremely important, resulting in the possibility of a comparison analysis. The functionality of scheduling events is also very beneficial to WSNs, which may have a number of nodes performing various operations in different layers simultaneously.

8.2 SWANS

SWANS is a high-performance WSN simulator that is based on JiST. SWANS was used to create the actual WSN network and communication routes between all nodes and clusters within the network. SWANS is mainly responsible for creating the acting objects within a WSN network (nodes, CHs, etc.) as well as defining their network layers. Nodes and CH objects are created by SWANS and simply exist within the simulation by being attached to the field, therefore existing within the field [3]. SWANS also breaks

down each node object into its various network layers: Radio (physical), MAC, Routing, Transport, and Application, creating an entity for each (see Figure 24 below).

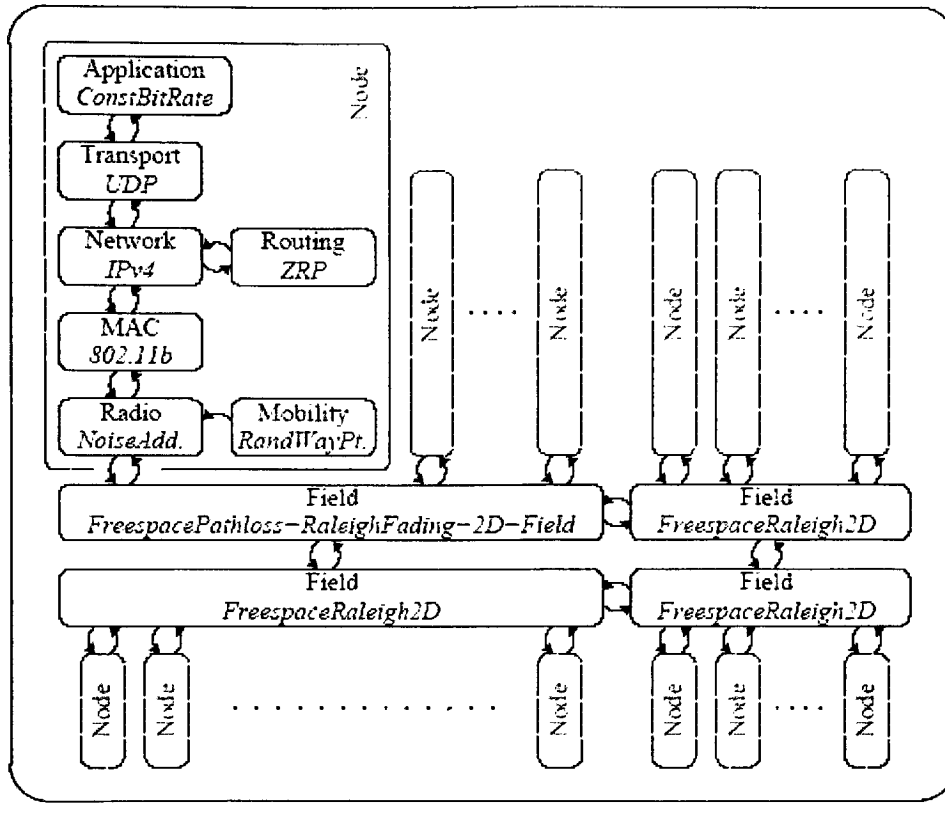

Figure 24: SWANS architecture [3]

SWANS does not natively have a function or layer that tracks energy. The tool also does not include generic protocols for each protocol layer it supports. Therefore, in order to conduct the simulations for SPECTRA, SWANS was reworked to support the necessary functions. A battery layer was added in order to keep track of energy during simulation runtime. In order to do this, the SWANS simulator was extensively modified because battery support needs to be present whenever simulation functions are conducted. Therefore, at every point of the simulation, energy depletion had to be supported. Along with the addition of a battery object, simple protocols were implemented for every

protocol layer that was needed for simulations. The implemented protocols were: a simple MAC protocol and an application layer protocol for both nodes and the BS.

All protocol layers within SWANS are implemented with a certain level of abstraction, which allows them to be completely independent. Since SPECTRA is a routing and security protocol it was implemented in the routing layer of the SWANS objects. In case certain layers are dependent on one another they can be associated by implementing the layers as "helper" classes. Another robust aspect of SWANS is the idea of both nodes and CH objects. In the case that the particular protocol being implemented does not use CHs and nodes, SWANS allows the programmer to ignore the CH object. This makes SWANS an extremely robust and practical engine to use for various WSN architectures [3].

SWANS is also ideal for WSN simulation since it is a high performance and scalable simulation engine. For every message transmitted within a WSN simulation, the simulator needs to determine all nodes that are affected by it, either by receiving the message, sensing it, or receiving interference from it. Most simulators do a linear search through all the available radios, resulting in a huge overhead and an extremely slow simulation run-time. Instead, SWANS breaks the field into quad tree representation (a hierarchal decomposition into squares), reducing the search effort for the affected nodes and receptors of a transmission [3].

8.3 Wireless Sensor Network implementation in JiST/SWANS

JiST and SWANS provide a good starting point for implementation of a WSN. JiST provides the core simulation engine, and SWANS implements both an efficient

Field for propagating messages and a complete network stack. However, certain design limitations in the base distribution of SWANS create challenges in developing a WSN simulator.

In order to support network layer interchangeability, SWANS defines a standard interface for each layer. This lends the extra advantage that the interface for each layer is not bound by the constraints placed on JiST entity classes. In fact, this is necessary so that method calls that cross layer interface boundaries can still be implemented. Because a class implementing one layer does not know what class will be implementing adjacent layers, only that class must implement the interface.

There are however, certain problems generated by the SWANS layer interface definitions. First, they represent a full-fledged IP stack, providing the application layer with sockets and allowing for multiple network interfaces (between the network layer and one or more MAC layers). In a WSN, such a powerful, general-purpose network stack is unnecessary. There is most likely only one application, one routing protocol, and one pair of MAC/Physical layers.

Another problem caused by the SWANS layer interfaces is that they name the specific types of addresses that various network layers use. The MAC layer is constrained to use 48-bit MAC addresses, and the Network layer uses 32-bit IP addresses. In a WSN, it is better for all layers to use a single type of address since pre-deployment procedures can guarantee that all the addresses are unique. Furthermore, depending on the size of the network, these addresses can easily be reduced to eight or 16 bits. SWANS also restricts the message types that can be passed across these interfaces, requiring full IP headers at the network layer, etc., and not allowing for additional fields easily.

Although useful in an IP stack, a network layer is unnecessary for determining which network interface to use. WSNs need only a routing protocol between the transport and MAC layer, and in this case a security protocol. Similarly, the transport layer itself (and the multiple ports and sockets it can provide to applications) is unnecessary. There are only a few applications for which it would be convenient to communicate directly with the network layer, and 100% end-to-end reliability of application messages is difficult.

Finally there are some features missing in SWANS that are necessary for a useful study of WSNs. It does not allow for the transmission strength of a transmitter to be changed after it is created, and there is no support for a battery model, or any form of power consumption tracking. SWANS was modified in order to support this functionality, as described in the following sections.

8.3.1 SPECTRA Layers

The layers in SPECTRA are organized as follows:

1. The Field (medium) is provided by SWANS.
2. The Radio forwards the received signal strength to the MAC and Net layers, and tracks power usage when transmitting and receiving. (Provided by SWANS)
3. The MAC Layer is a simple CSMA scheme that is provided by SWANS.
4. The Network layer combines the functionality of a routing layer and the security protocol. (Implemented)

5. Two application layers were written: one for the BS (Sink), and one for the remainder of the nodes, including CHs. (Implemented)

8.3.1.1 MAC Layer

The MAC layer is responsible for sending messages, avoiding collisions, and guaranteeing that packets are transmitted for one hop. It implements a simple CSMA collision avoidance protocol. When sending a message, the MAC layer waits for a random time up to 20 bytes (0.16 ms). This spreads out messages which would otherwise be occurring at the exact same time, such as responses to a broadcast. Then the MAC layer checks the current state of the radio to see if it is OK to transmit. If not, then the MAC layer backs off for a random time between 1000 and 1300 characters (8-10.4 ms) before checking again. This delay is unnecessarily large, but high performance is not needed and fine-tuning the MAC layer is beyond the scope of this work.

Once the MAC layer detects that the channel is free, it sends the message and begins waiting for an acknowledgement packet. If it receives an *ACK* signals the network layer that it is free to transmit the next packet. If it does not receive an *ACK* within 40 ms, it tries to resend, backing off as necessary while the channel is blocked.

8.3.1.2 Routing/Security Layer

The routing and security layer is the foundation of SPECTRA. All security is provided at this layer, in a fashion that is transparent to the application layer. All of the keys are held in the security layer of SPECTRA. Information about a nodes position and role in the SPECTRA system, such as lists of neighbors, if the node is a CH, and what

nodes are in the same cluster are held in the routing layer. The routing layer insures that the system organizes itself correctly into clusters, and establishes routes. It keeps track of routing tables, and deals with all messages related to the operation of the system and message routing.

8.3.1.3 Application Layer

There are two Application layer classes, one for the BS (BSApp) and one for the remainder of the nodes (NodeApp). NodeApp thus handles both CHs and ordinary sensor nodes.

8.3.1.3.1 Base-station

The BS application layer is the originator of data requests, which are directed to a single CH. It is also responsible for periodically instructing its network layer to broadcast a new *system key*.

8.3.1.3.2 Clusterhead

At the application layer, a CH is responsible for reporting aggregated data to the BS, both periodically and to fulfill a specific request, and requesting and aggregating data from its cluster's nodes. It also forwards node death messages from ordinary nodes to the BS.

8.3.1.3.3 Node

At the application layer, an ordinary node is only responsible for generating periodic data traffic, responses to specific requests, and informing its CH if it is low on power and about to die.

8.3.2 Event entry points

In order to provide a road map for the source code documentation, a list of events that may be scheduled during the course of the simulation is given. For each event, the scheduling circumstances and responsibilities are as stated. All major events/functions implemented for each layer can be seen in Tables 3-7. A detailed description of each function can be found in the appendix.

Field	
Function Name	**Functionality/Use**
void transmit()	Creates a transmission in the field and determines nodes in the field
Radio	
void setPower()	Adjusts the transmitting power and updates the node battery object
void transmit()	Initiates transmission, simulates delay, and calculates energy usage
void endTransmit()	Resets radio state to minimize power usage
void receive()	Turns on at the beginning of transmit, calculates power usage for receive
void endReceive()	End receive and calculates power

Table 3: Field Functions

MAC	
Function Name	**Functionality/Use**
setRadioMode()	Called to update MAC layer on radio's current state
void receive()	Called when a receive has occurred. MAC layer determines destination
void setRadioPower()	Changes transmission strength
void acknowledge()	Called to send an *ACK* to sender
void sendMessage()	Performs back-offs, sends the message if the medium is clear, and waits for acknowledgement
void forgetMessage()	Invoked after a specific delay to clear messages in the received list
void send()	Called by network layer to send any packet other than an acknowledge

Table 4: MAC Layer Functions

Network Layer	
Function Name	**Functionality/Use**
void receive()	Called by the application layer when receiving a message. Responsible for decrypting incoming messages.
void send()	Called by the application layer when a message is being sent. Responsible for encrypting outgoing messages.
void pump()	Retrieves next message that the routing layer is waiting to transmit
void broadcastCHAuthenticationRequest()	Sends initial *authentication request* that every CH must send to the BS
void broadcastAuthenticationRequest()	Sends initial *authentication request* that every node must send to the BS
void broadcast*RREQ*()	Called when a route needs to be established to some destination
void broadcastClusterJoin()	This message joins a node to a cluster and notifies its neighbor nodes
void broadcastClusterAdvertisement()	CHs send this message to nodes informing them they can join its cluster
void broadcastNodeRemoval()	Message is sent by a node that discovers one of its neighbor nodes is missing
void joinCluster()	Determines which cluster to join based on signal strengths
void start()	Initiates first authentication attempt at system startup
void refreshSystemKey()	Initiates a refresh of the *system key*
void unableToContactNeighbor()	Notifies that a neighbor has failed to respond 5 re-try attempts and initiates a removal

Table 5: Network Layer Functions

Node Application Layer	
Function Name	**Functionality/Use**
void receive()	Receive is called in response to a message received by the app layer
void send_aggregated_data()	Event scheduled after the first node responds to a request
void clear_request()	Occurs after a period of time after a send_aggregated. Clears late replies
void check_battery()	Checks battery level, if less than 5% message is sent
void run()	Starts nodes periodic operations and start up actions.

Table 6: Node Application Layer Functions

BS Application Layer	
Function Name	**Functionality/Use**
void requestLoop()	Sends a request at random to a CH for its aggregated data
void systemKeyLoop()	Periodically instructs the network layer to update the *system key* in all nodes

Table 7: BS Application Layer Functions

8.3.3 Message types

The messages are built from top to bottom by appending the current layer's header to the message being passed down, and decomposed by the reverse process. Thus the simplest message is one created by the MAC layer, and the most complex is one created by the application layer. For uniformity and simplicity of design, all packet fields are 16 bit signed integers to match the address size. All messages implemented for each simulation layer can be found in Tables 6 and 7. A detailed description of each message can be found in the appendix.

Net Layer Messages	
Function Name	**Functionality/Use**
ROUT_*RREQ*	Used to establish a new route in the network
ROUT_*RREP*	Convey the new route back to the requesting node, eavesdropped by other nodes
APP_MSG	Primarily used for data transmission and aggregation
APP_MSG_BCAST	Same as APP_MSG except broadcast to all nodes instead of a particular node
CMD_FIRST_GROUP_KEY	Reply sent by BS in response to *authentication requests*.
CMD_REFRESH_GROUP_KEY	Primary method of continuous authentication within the SPECTRA network.
CMD_NEW_CLUSTER_KEY	Sent whenever a cluster needs to distribute a new *cluster key* to its nodes. Used to refresh the *cluster key* after a node in the cluster has been compromised.
CLUSTER_ADVERTISEMENT	Nodes record the RSS with which they receive this message, and join the CH from which they received the highest RSS.
CLUSTER_NODE_JOINS	Informs the CH that the node is joining the cluster.
SEC_NODE	Initial request for authentication from a node to the BS. BS replies with a CMD_FIRST_GROUP_KEY message.
SEC_CH	Initial request for authentication from a CH to the BS. BS replies with a CMD_FIRST_GROUP_KEY message.
NODE_REMOVAL	This message informs all recipients to remove the missing node from their routing tables, and if a CH receives this message, it will refresh its *cluster key* because the old *cluster key* may be compromised.
CLUSTER_REORG	This command is sent by the BS when it discovers that one of it's CHs is gone. The nodes that used to belong to the now missing CH must join nearby clusters.
REFRESH_CLUSTER_KEY	Called by the CH when it discovers that one of its nodes is missing and has perhaps been compromised.

Table 8: Net Layer Messages

Application Layer Messages	
Function Name	**Functionality/Use**
short NODE_DATA = 100	Data sent from a node to its CH on a periodic basis, not associated with a specific BS request.
short CH_DATA = 101	Aggregated data sent from a CH to the BS, not associated with a particular request.
short NODE_RESPONSE =102	Data sent from a node to its CH in response to a specific request.
short CH_RESPONSE = 103	Aggregated data sent from the CH to the BS in response to a specific request.
short CH_REQUEST = 104	Request for data from the BS to a CH.
short NODE_REQUEST = 105	Request for data from a CH to one or all of its nodes.
short BEGIN_MONITOR =106	Instruction for a node or CH to start making periodic reports of data.
short END_MONITOR = 107	Instruction for a node or CH to stop making periodic reports of data.
short NODE_DEATH =108	Notification that a node is low on power.

Table 9: Application Layer Messages

Chapter 9. Security Performance Analysis

The feasibility of a proposed protocol cannot be determined until some sort of analysis has been conducted. In the case of SPECTRA, a simulation analysis of the protocol was done in order to determine its validity and benefits. The topology of the network in the simulations is dynamic and determined at run time. In order to reduce this variability the simulations were run with the same parameters 100 times, and the results averaged. Simulations were conducted by varying only one metric in order to compare the results and reduce the effect of other aspects. The following metrics were used for protocol performance analysis: number of nodes and CHs, battery size (initial battery size), average energy depletion, packet size, encryption levels, and the overhead of encryption levels. A detailed description of the metrics can be seen below.

- **Number of nodes and CHs**: the effect on the network lifetime as nodes are added and CHs are added. (how scalable is the network, can either nodes and CHs be added without adding the other type)
- **Battery size**: the effect of the initial battery size on network lifetime. (do both CH and node batteries need to be increased together)
- **Average energy depletion**: how energy is consumed over the course of the network lifetime. (where is energy consumed more)
- **Packet size**: how scalable is the protocol for other applications where larger packets are required.
- **Encryption levels**: how different levels of encryption effect the overall network lifetime. (how much of a difference is there between 64 bit and 128 bit encryption)

- **Overhead of encryption levels**: what is computational intensity of encryption and its effect on the network.

As mentioned in Chapter 2, this work shows the benefits of the proposed protocol by showing the reduction in energy consumption through simulations. Because there is a lack of current work in this particular technique, a comparative analysis is extremely difficult. However, the energy consumption of traditional protocols that do not integrate routing and security can be compared. The following tables show the energy consumption of the SPINS Security Protocol [30], the Pairwise security protocol [30], and the Pebblenets protocol [30] utilizing the same battery as seen in this work[3].

Consumption Type	Node Energy (mW)	System Lifetime (hours)
Communication	128	3.5
Computation	23	
Total	151	

Table 10: SPINS Protocol Energy Consumption [30]

Consumption Type	Node Energy (mW)	System Lifetime (hours)
Communication	236	.5
Computation	36	
Total	272	

Table 11: Pairwise Security Protocol Energy Consumption [30]

Consumption Type	Node Energy (mW)	System Lifetime (hours)
Communication	31	7.5
Computation	1.63	
Total	32.63	

Table 12: Pebblenets Protocol Energy Consumption [30]

[3] Assuming a CR2032 watch battery with an output of 300mAh

9.1 Results and Analysis

An extremely important aspect of any security protocol is the effect it has on energy depletion of a network. It is a greater issue in the SPECTRA protocol because the energy depletion of the network decides the difference between the initial battery size of a CH and node. Figure 27 shows the average energy depletion of nodes and CHs over time in a given simulation. The topography of the network can be seen in Figure 25, while the backbone of the network can be seen in Figure 26. Figures 24 and 25 are examples of what a typical WSN will look like for the following simulations. In the beginning and in the end of Figure 27, the slopes are greater when compared to the center part. This is because routing and *system setup* is being conducted in the beginning, and therefore more messages are passed between nodes and CHs. In the end, more energy is expelled because nodes and CHs are dying out, and therefore low battery messages are being sent as well as rerouting into other clusters.

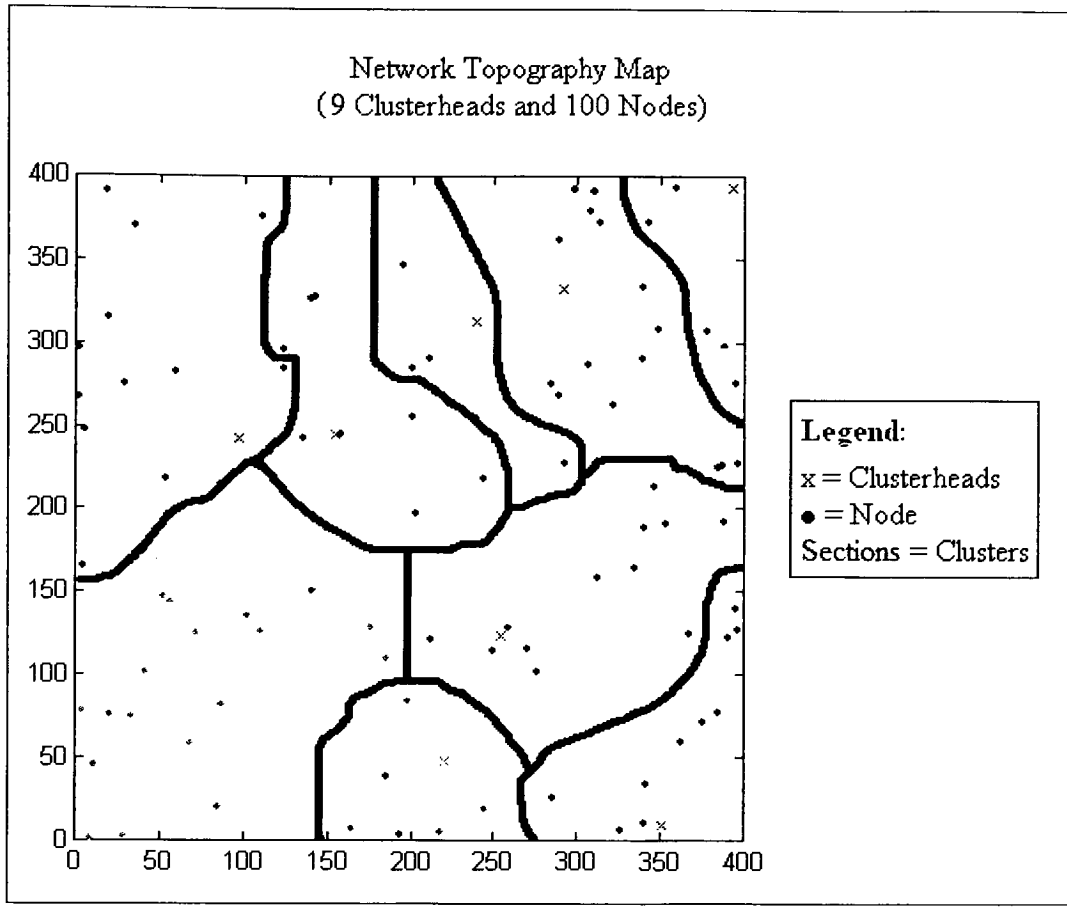

Figure 25: Network Topography Map

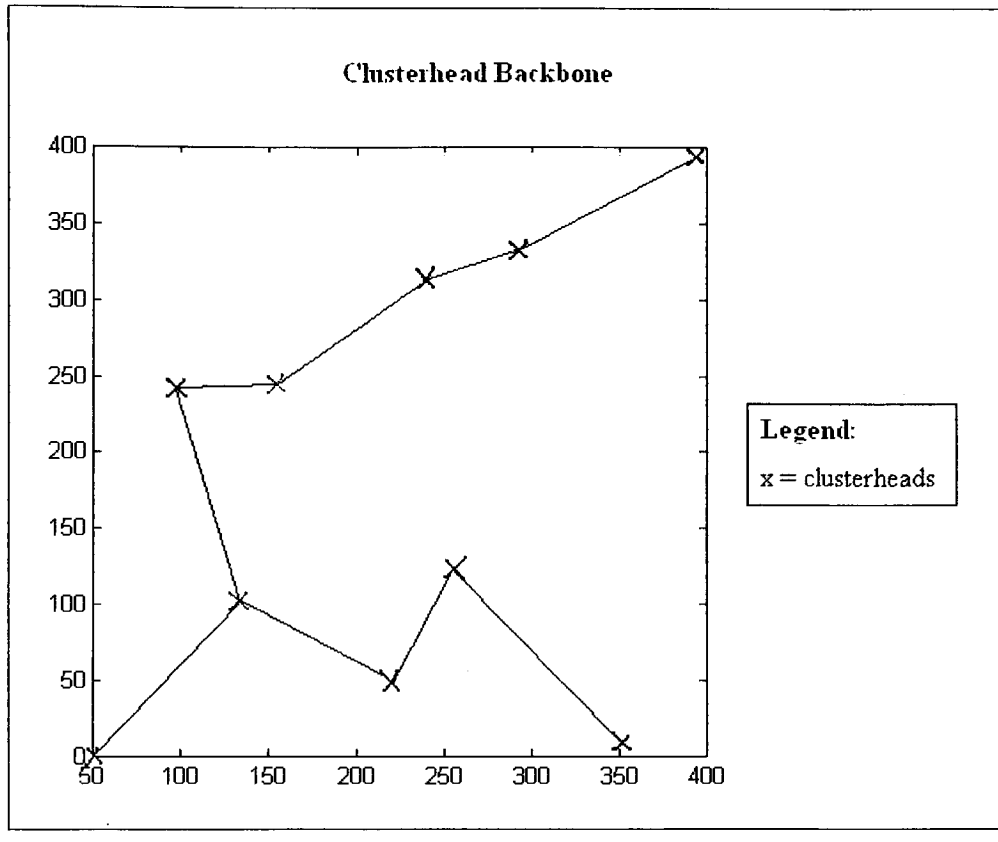

Figure 26: CH Backbone

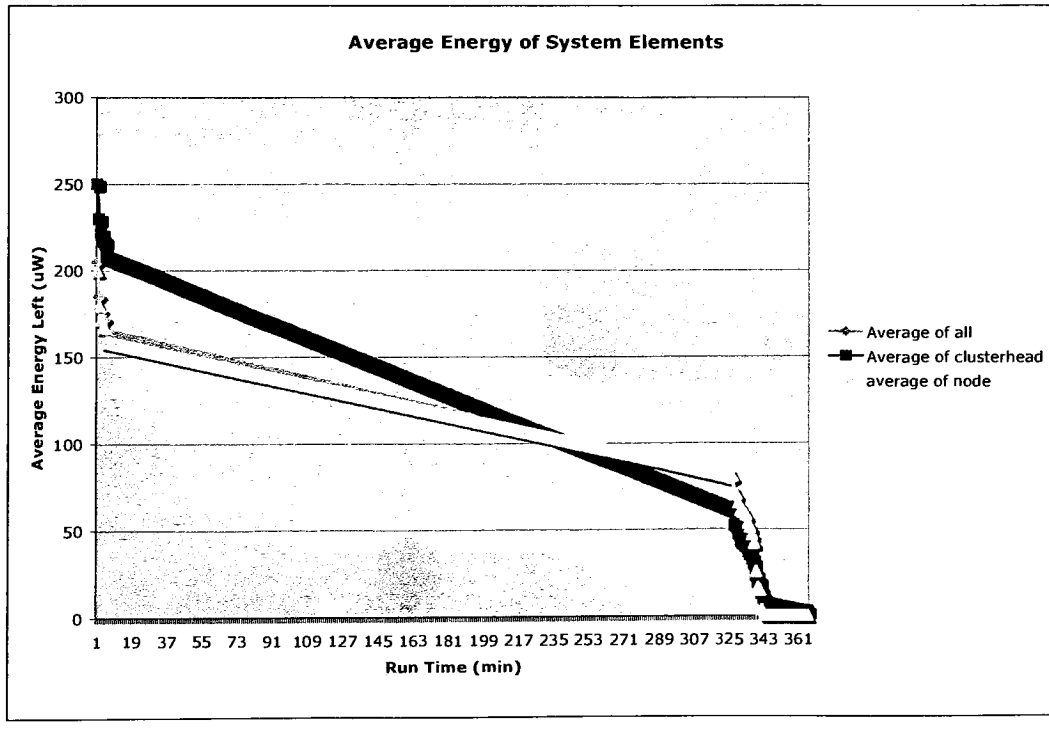

Figure 27: Average energy of system elements over run time

It is important to understand how the network size in terms of the ratio of CHs to nodes effects the system lifetime. Figure 28 shows the effect nodes have on the overall lifetime of a given SPECTRA network. In this particular case, CHs were kept constant while the number of nodes was varied. As nodes increase, the network ratio changes and the overall system lifetime decreases. With more nodes in the system, the overall routing and setup time of the system increases, which is the area where energy is depleted faster (mainly the beginning and end parts of Figure 27 are increased). With more nodes, cluster sizes are increased (as seen by the lowered ratio), creating a larger overhead for the average CH. Therefore, when nodes in a system are increased after a certain threshold system lifetime decreases.

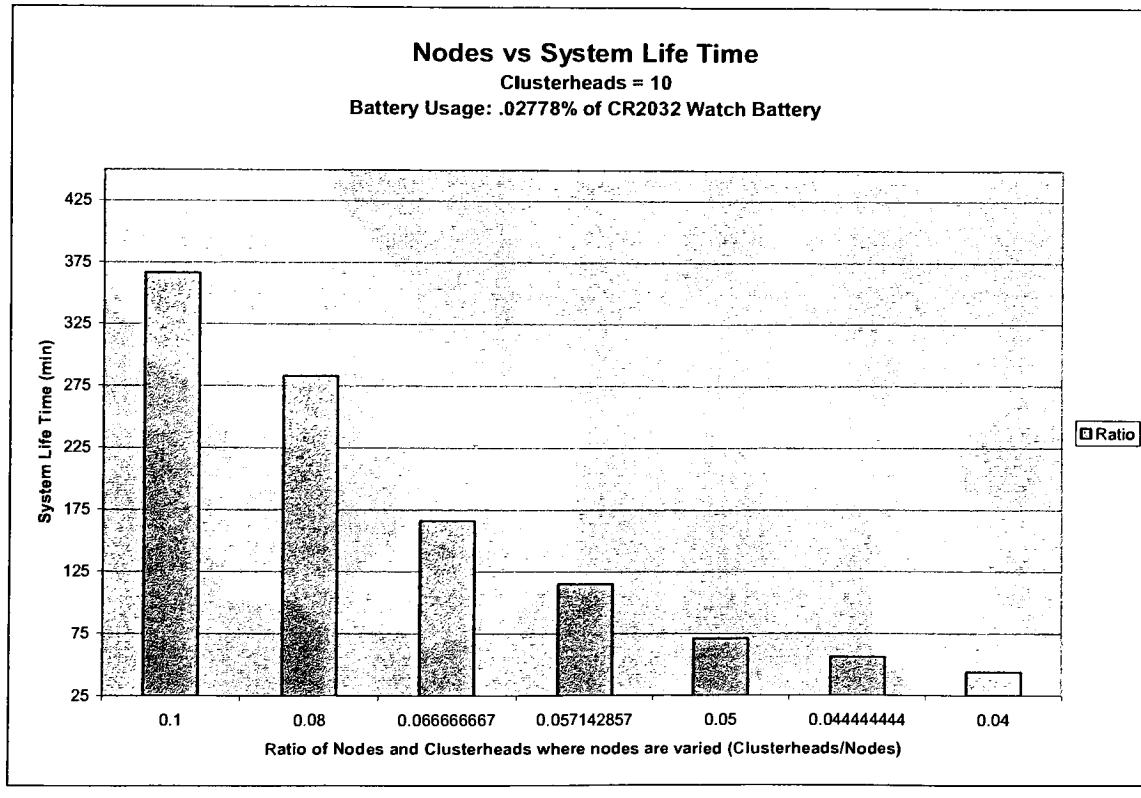

Figure 28: Change in overall system lifetime due to the number of nodes (varied ratio)

The figure below shows the effect of CHs on the lifetime of a given SPECTRA system. In this particular simulation, the nodes are kept constant while the number of CHs is varied (resulting in an increased network ratio). As demonstrated, there is an equilibrium number of CHs that should be in the system. This is because after a certain number of CHs, additional CHs cause problems because clusters are too small, data isn't being correlated anymore, and nodes don't have multi-hop paths to their CH. As a result, nodes expend more energy communicating with their CHs, and CHs communicate more with the BS because aggregation is more frequent. As can be seen in the figure below, it is likely that the highest point of the trend is the optimum ratio for a network consisting of a given number of nodes and clusterheads. Figure 29 shows that a favorable number of CHs for a network consisting of 100 nodes is 10 (network ratio of .1).

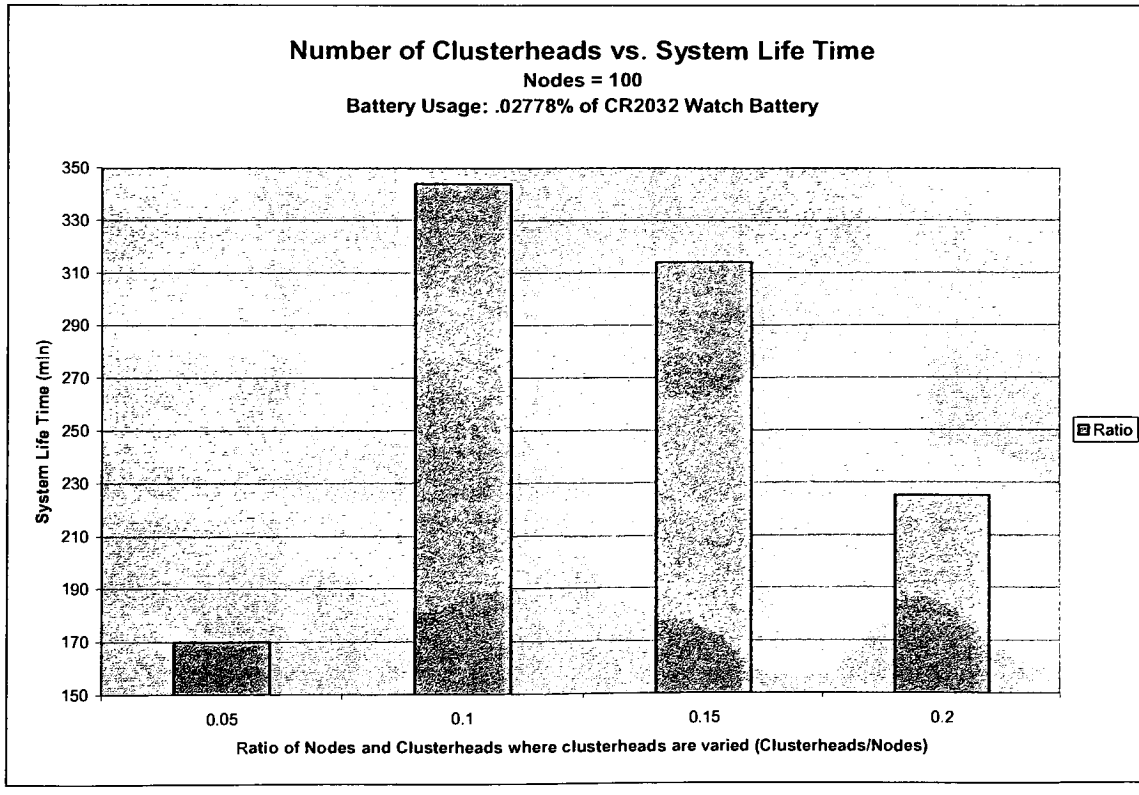

Figure 29: Change in overall system lifetime due to the number of CHs (varied ratio)

The lifetime of a given entity is dependent on the battery utilized. Larger batteries result in an increased system and node lifetime. However, when a larger battery is utilized for a given entity in the system, a larger battery will also be needed for other parts of the system. It can be seen in Figure 30 and Figure 31 that when either the node or CH battery is held constant, the variable battery can only be increased to a certain level before it is useless. In the case where the CH's battery is kept constant, the node battery can increase only to the point where the CH becomes the limiting factor in the lifetime. The same situation occurs as the CH battery is varied, and the nodes' kept constant. After a certain threshold, the system dies because the nodes lack power. Therefore, it is important to keep a ratio that increases the system lifetime and keeps the dependency of lifetime not on the entities, but rather the system topography.

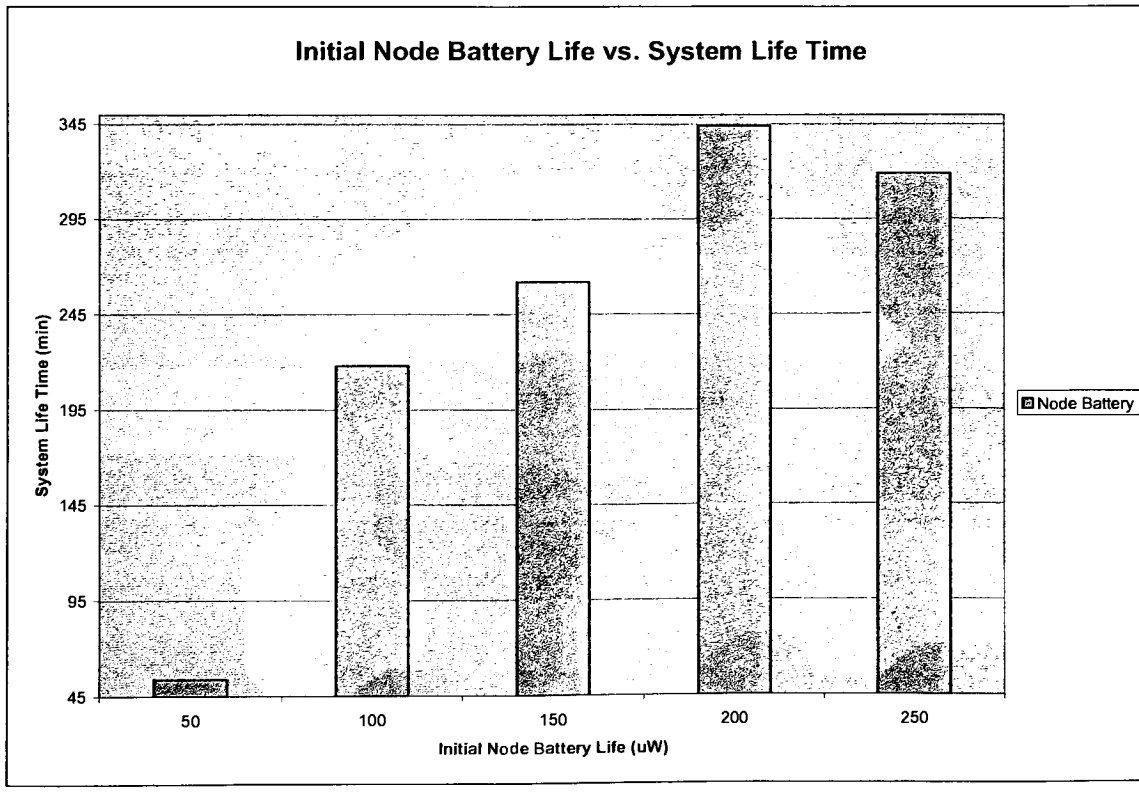

Figure 30: The effect of a node's initial battery level on network lifetime

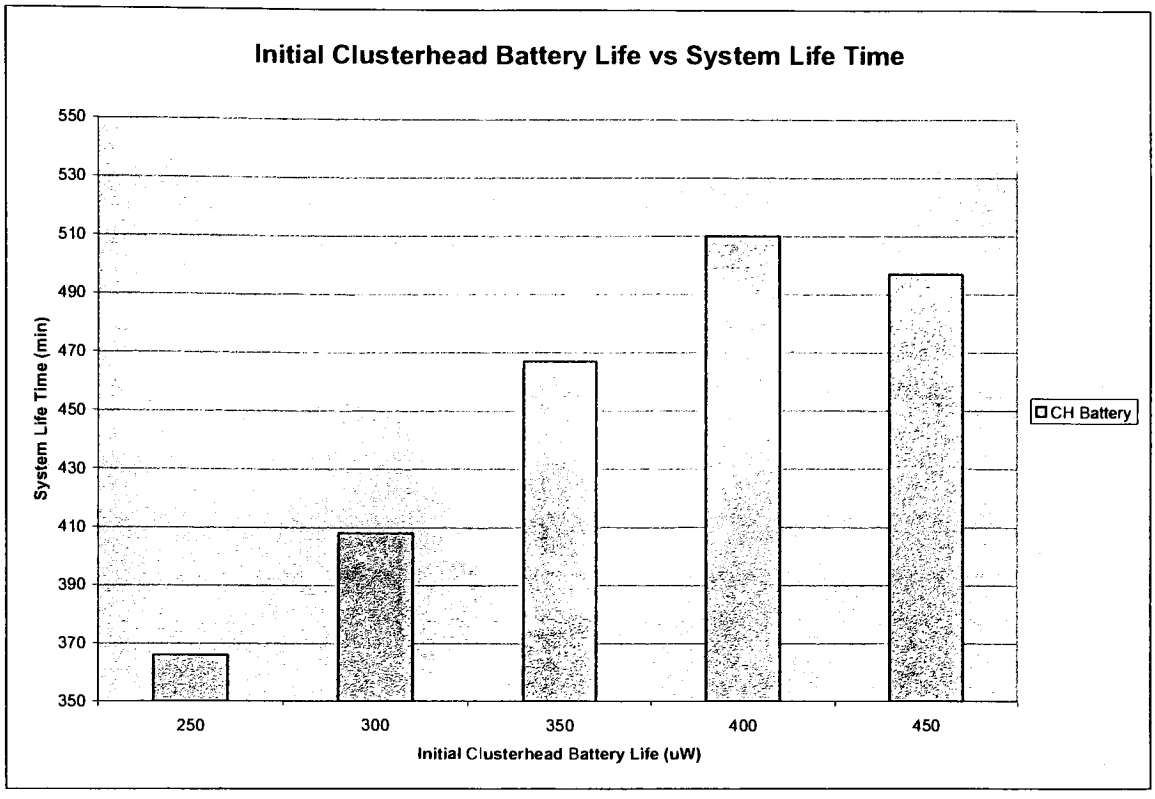

Figure 31: The effect of a CH's initial battery level on network lifetime

The packet size within a system greatly affects network lifetime. A larger packet size increases the information being transmitted, resulting in a longer on state for the transceiver. As packet sizes increase, overall system lifetime decreases, as can be seen in Figure 32 below.

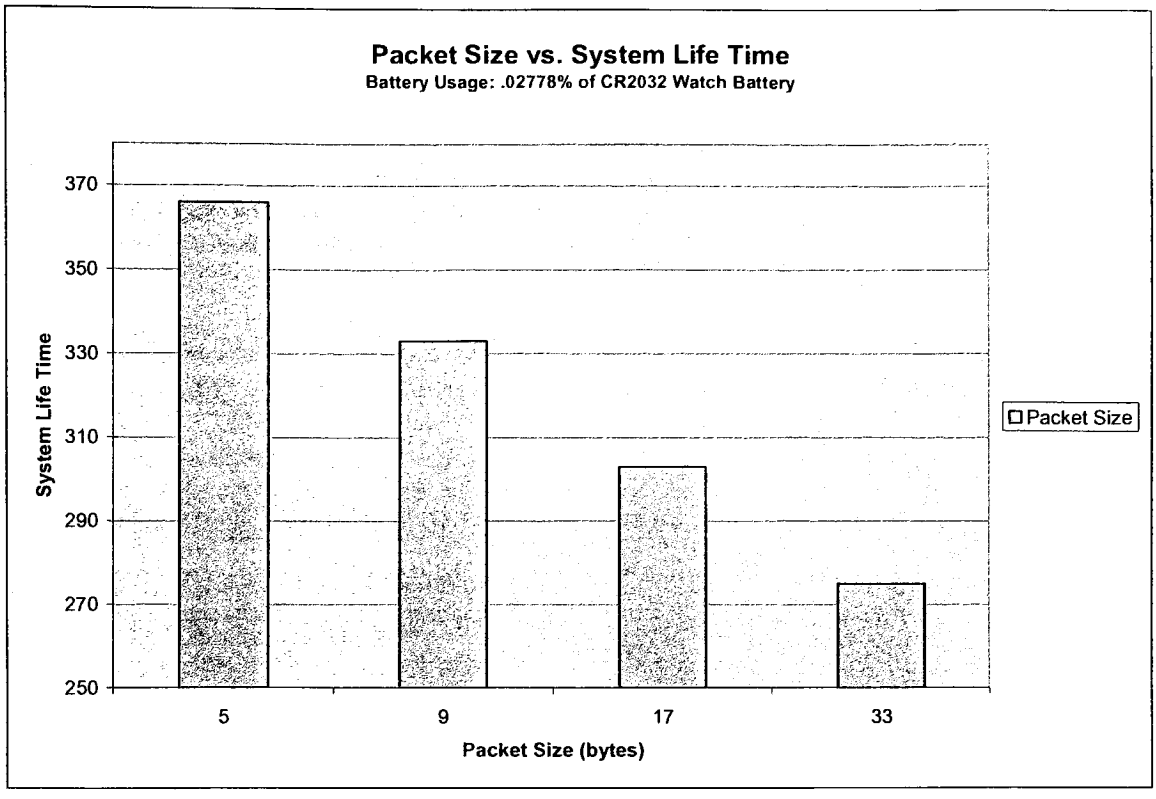

Figure 32: The effect of packet sizes on network lifetime

The most important effect on system lifetime of a security algorithm is the effect that encryption has on power consumption. As can be seen in Figure 33 (below), the effect of encryption on power consumption is exponential. Increases in the level of encryption double the power consumption required for encrypting a packet. In other words, 64 bit encryption requires twice as much power as 32 bit encryption. Power usage increases drastically because encryption is used at every point in the SPECTRA network. It is important to note the amount of power consumption related to the battery capacity. As can be seen by Figure 33, the amount of power consumed for encryption is fairly minimal compared to the overall battery capacity. Therefore, even though higher levels of encryption require far more power, the power consumed by encryption is far less than the power consumed by communication overhead.

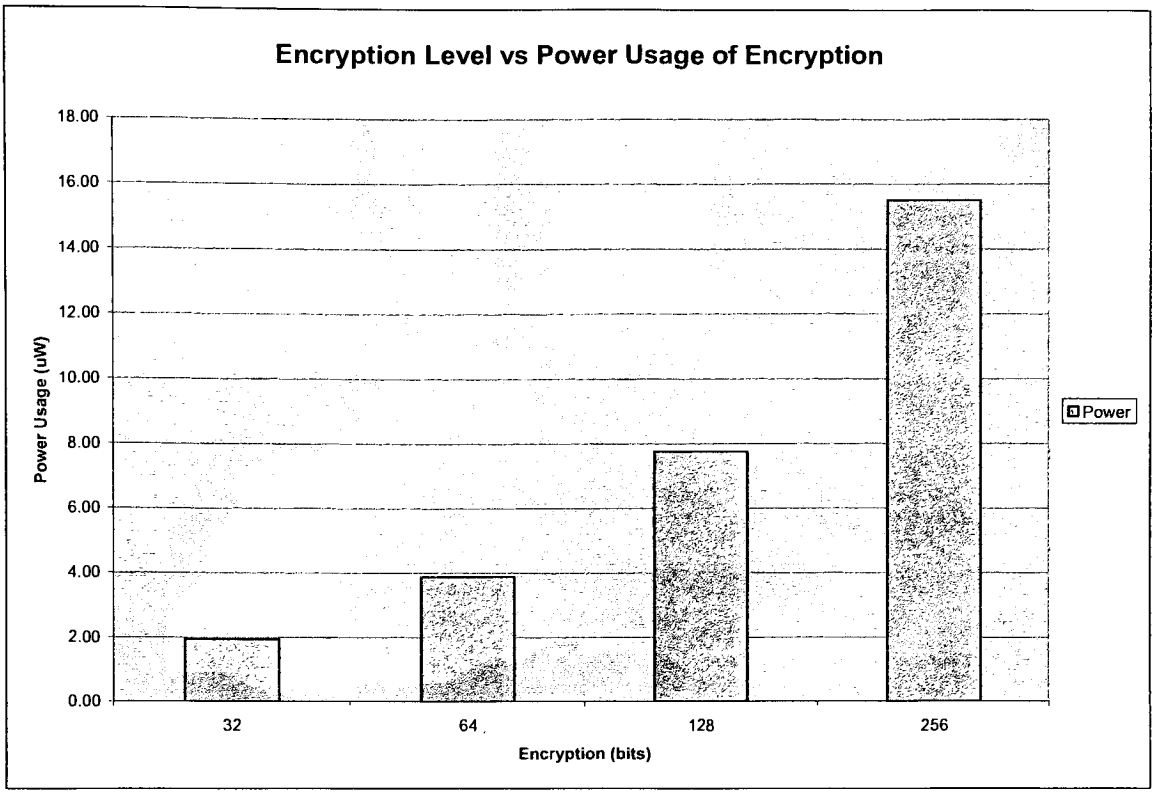

Figure 33: Power consumption associated with different encryption levels

The effect encryption has on overall system lifetime is actually greater than doubling. This is caused by encryption being used in every aspect of the system, both in decrypting and encrypting any kind of message. Therefore, the encryption of a particular packet doubles, but the overall system lifetime decreases by more than half. This can be seen in Figure 34 below.

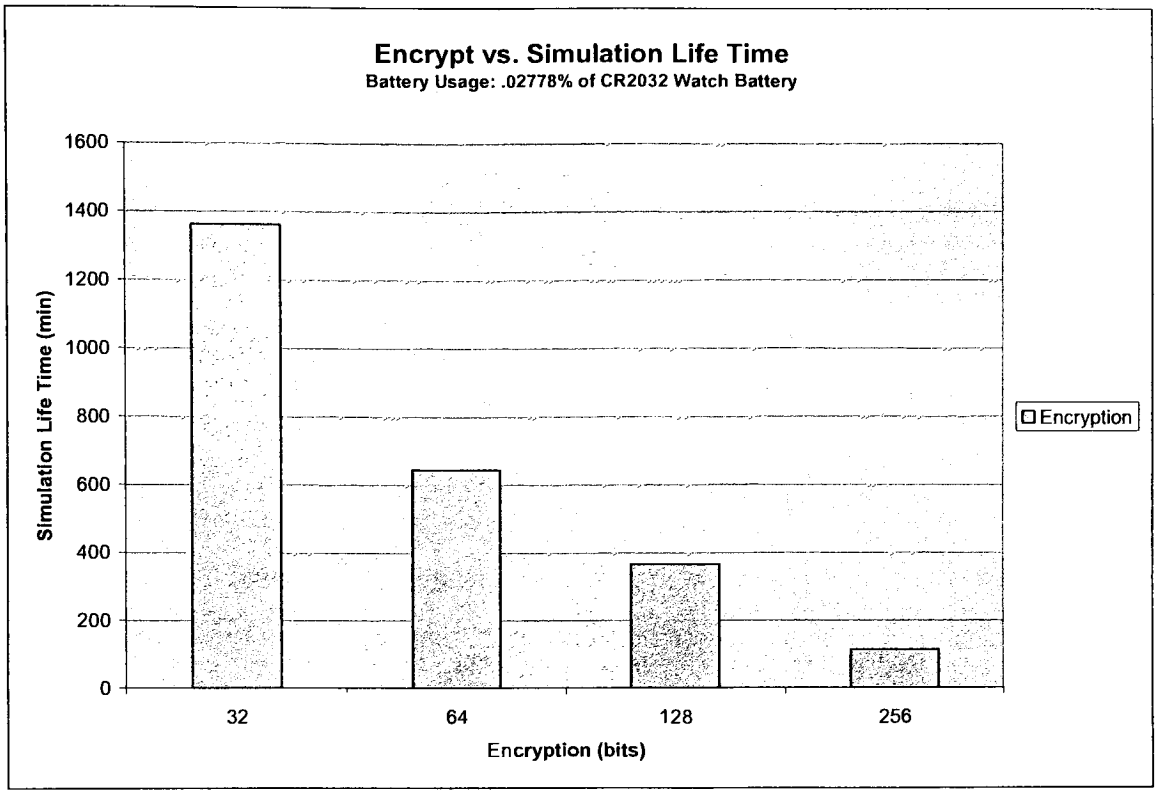

Figure 34: The effect of encryption on network lifetime

Analysis of the results show the practicality of the SPECTRA protocol for WSN applications. Traditional security protocols that do not integrate routing and security run for a few hours utilizing mWs of power (Tables 10-12 [30]). SPECTRA on the other hand is extremely power efficient, running for hours on a few μW of power. The results also show the scalable and adaptive nature of SPECTRA. SPECTRA can employ various levels of encryption with little effect on power consumption. It can easily be scaled for different network sizes without drastically varying power consumption. SPECTRA is also adaptive in network communication procedures, allowing for different packet sizes. This allows the protocol to be implemented in various applications. The results prove that SPECTRA is an effective security protocol that achieves scalability, adaptive security, routing, all at a low cost to power.

Chapter 10. Conclusions and Future Work

One of the most challenging topics in WSNs is security due to the ad hoc nature, intermittent connectivity, and resource limitations of WSNs. Current solutions to the security issue in WSNs were created with only authentication and confidentiality in mind. This is far from optimal, because routing and security are closely correlated. Routing and security are alike because similar steps are taken in order to achieve these functions within a given network. Therefore, security and routing can be combined together in a cross-layer design, reducing the consumption of resources.

This work integrated routing and key management in order to provide an energy efficient security and routing solution. To achieve this, the following features were implemented: integration of security and routing, dynamic security, robust re-keying, low-complexity, and dual levels of encryption. This work combined all the robust features of current security implementations while adding additional features like dual layer encryption, resulting in an extremely efficient security protocol. SPECTRA was designed to provide all of these features while decreasing the effect on network lifetime by its low power nature.

Aggressive analysis of the simulation results shows the practicality of the SPECTRA protocol and its low operating cost in power consumption. The simulation results show the lifetime of a given SPECTRA network in hours when utilizing only 2 hundredths percent of a watch battery[4] (a few μW of power). Given the lifetime of the network on a small percentage of power that will be allocated in actuality, shows the potential lifetime of the network. The power overhead associated with SPECTRA is

[4] Assuming a CR2032 watch battery with an output of 300mAh

relatively little when compared to nonintegrated approaches to routing and security (Tables 10-12), making SPECTRA extremely power efficient and increasing network lifetime.

Packet size has a large effect on the network lifetime because a WSN is based on data transmission and reception. Larger packets require larger transmission and receiving times, resulting in a noticeable effect on network lifetime. Therefore, it is pertinent to choose the size of a system packet carefully. The smallest packet that can be utilized to get the network functions done is most useful because it increases network lifetime.

The cross-layered approach to the SPECTRA protocol benefits a network by decreasing energy consumption, increasing network lifetime. A cross-layered approach also reduces the number of protocols needed and combines communication to benefit multiple layers. This is achieved by conducting routing and security set up simultaneously, combining the two layers and the communication needed. This reduces power consumption and superfluous communication for routing and security.

It is important to understand that security in SPECTRA is changeable or flexible. This allows a specific network to implement higher security levels, through bigger keys, if the application requires it. As the results showed, system lifetime degrades as security is increased, due to the overhead associated with encrypting and decrypting larger keys.

Though larger keys add to the computation overhead, this overhead is small when compared to network communication. Transmission and reception of data packets consume far more power then computations. Though it is important to understand computation overhead in a security protocol is seen over and over again because encryption is used on every packet. Therefore, not only must a packet be encrypted but

also decrypted. In the end, larger keys do affect power but the effective consumption is small when compared to transmission/reception consumption. It is important to use the level of encryption that is required for the environment that the network will be in.

SPECTRA has been proven to be an effective security protocol that achieves scalability, adaptive security, routing, all at a low cost to power. The adaptive security can be seen in the level of encryption and in the ability to adapt to network topology changes. SPECTRA allows for a network to run for days on less than 1 percent of the battery capacity[5] (a few μW of power), where traditional implementations run for hours on mW of power (Tables 10-12 [30]). SPECTRA is robust in its implementation, features, and utility. The protocol is ideal for low power, adaptive networks that require a high level of security.

The next step to take in the validation of the SPECTRA protocol is to implement it in hardware. Simulation can only provide an estimated guess of how the protocol works in actuality. In order to completely understand the effect of the protocol on a network, it needs to be implemented in an actual application and in actual hardware. Hardware implementation also provides aspects that cannot be modeled in simulation such as environmental aspects or mobility. With hardware implementation, the effect that other variables have on the protocol can be analyzed. Along with a hardware implementation, it will be useful to understand how SPECTRA works with protocols from different layers.

The SPECTRA protocol is only for a specific layer in the protocol stack. As a result, other protocols will be implemented for the other layers. It is very likely that the bottleneck of a network could exist in one of the other layers due to the protocol chosen.

[5] Assuming a CR2032 watch battery with an output of 300mAh

Therefore, SPECTRA should be analyzed with protocols from other layers in order to determine if its performance degrades with particular protocols in comparison to others.

Bibliography

[1] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. "SPINS: Security Protocols for Sensor Networks." *University of California, Berkeley.* <http://www.ece.cmu.edu/~adrian/projects/mc2001/mc2001.pdf>.

[2] Barr, Rimon. "JiST - Java in Simulation Time Users Guide." *Cornell University,* March 19, 2004 <http://jist.ece.cornell.edu/docs/040319-jist-user.pdf>

[3] Barr, Rimon. "SWANS - Scalable Wireless Ad hoc Network Simulator Users Guide." *Cornell University,* March 19, 2004 <http://jist.ece.cornell.edu/docs/040319-swans-user.pdf>.

[4] Stefano Basagni, Kris Herrin, Danilo Bruschi, and Emilia Rosti. "Secure Pebblenets." *University of Texas, Dallas.* <http://www.cs.huji.ac.il/labs/danss/sensor/adhoc/basagani_2001securepebblenets.pdf>.

[5] RSA Laboratories. "What is a hash function?" RSA Security Section 2.1.6. <http://www.rsasecurity.com/rsalabs/node.asp?id=2176>.

[6] Adrian Perrig, Robert Szewczyk, Victor Wen, Alec Woo, "Security for SmartDust Sensor Network," the whole paper is available from the following website: <http://www.cs.berkeley.edu/~vwen/classes/f2000/cs261/project/sensor_security.html>.

[7] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," Proceedings of Seventh Annual International Conference on Mobile Computing and Networks MOBICOM 2001, July 2001.

[8] Donggang Liu and Peng Ning, "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks," Proceedings of the 10th Annual Network and Distributed System Security Symposium, pages 263 - 276, San Diego, California. February 2003.

[9] Donggang Liu, Peng Ning, "Multi-Level u-TESLA, A Broadcast Authentication System for Distributed Sensor Networks," Submitted for journal publication. Also available as Technical Report, TR-2003-08, North Carolina State University, Department of Computer Science, March 2003.

[10] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod Varshney, "A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks," Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), Washington DC, October 27-31, 2003.

[11] Haowen Chan, Adrian Perrig, and Dawn Song, "Random Key Predistribution Schemes for Sensor Networks," IEEE Symposium on Research in Security and Privacy, 2003.

[12] Roberto Di Pietro, Luigi V. Mancini, and Alessandro Mei, "Random Key Assignment for Secure Wireless Sensor Networks," 2003 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03) October 31, 2003 George W. Johnson Center at George Mason University, Fairfax, VA, USA.

[13] Laurent Eschenauer, Virgil D. Gligor. "A key-management scheme for distributed sensor networks." Conference on Computer and Communications Security". Proceedings of the 9th ACM conference on Computer and communications security 2002 , Washington, DC, USA

[14] Donggang Liu and Peng Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," The 10th ACM Conference on Computer and Communications Security (CCS '03), Washington D.C., October, 2003

[15] Donggang Liu and Peng Ning, "Location-Based Pairwise Key Establishments for Relatively Static Sensor Networks," 2003 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03) October 31, 2003 George W. Johnson Center at George Mason University, Fairfax, VA, USA.

[16] Wenliang Du, Jing Deng, Yunghsiang S. Han, Shigang Chen and Pramod Varshney, "A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge," To appear in IEEE INFOCOM'04, March 7-11, 2004, Hongkong.

[17] J. Deng, R. Han, and S. Mishra, "INSENS: Intrusion-tolerant routing in wireless Sensor Networks," Technical Report CU-CS-939-02, Department of Computer Science, University of Colorado, November 2002.

[18] Sarjoun Doumit, and Dharma P. Agrawal, "Self-Organized Criticality and Stochastic Learning-Based Intrusion Detection System for Wireless Sensor Networks," MILCOM 2003, October, 2003

[19] Jing Deng, Richard Han, and Shivakant Mishra, "A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks," 2nd International Workshop on Information Processing in Sensor Networks (IPSN '03), Palo Alto, CA, USA, April, 2003.

[20] Jing Deng, Richard Han, and Shivakant Mishra, "Security Support for In-Network Processing in Wireless Sensor Networks," ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03) October 31, 2003 George W. Johnson Center at George Mason University, Fairfax, VA, USA.

[21] Anthony D. Wood, and John A. Stankovic, "Denial of Service in Sensor Networks," IEEE Computer, 35(10):54-62, 2002

[22] Chris Karlof and David Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," First IEEE International Workshop on Sensor Network Protocols and Applications, May 2003.

[23] Guiling Wang , Wensheng Zhang , Guohong Cao , and Tom La Porta, "On Supporting Distributed Collaboration in Sensor Networks," MILCOM 2003, October, 2003.

[24] J. Zachary, "A Decentralized Approach to Secure Group Membership Testing in Distributed Sensor Networks," MILCOM 2003, October, 2003.

[25] S. Zhu, S. Setia and S. Jajodia. "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks." 10th ACM Conference on Computer and Communications Security (CCS '03), Washington D.C., October, 2003.

[26] Yee Wei Law, Sandro Etalle, Pieter H. Hartel, "Key Management with Group-Wise Pre-Deployed Keying and Secret Sharing Pre-Deployed Keying", EYES project (Europe), available through GOOGLE search engine.

[27] Sasha Slijepcevic, Miodrag Potkonjak,Vlasios Tsiatsis, Scott Zimbeck, and Mani B. Srivastava, "On communication Security in Wireless Ad-Hoc Sensor Network," Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02) June 10 - 12, 2002 Pittsburgh, Pennsylvania, USA.

[28] Mike Chen, Weidong Cui, Victor Wen, and Alec Woo, "Security and Deployment Issues in a Sensor Network," downloadable from <http://citeseer.nj.nec.com/chen00security.html>.

[29] Y.W. Law, S. Dulman, S. Etalle and P. Havinga, "Assessing Security-Critical Energy-Efficient Sensor Networks," Department of Computer Science, University of Twente, Technical Report TR-CTIT-02-18, Jul 2002.

[30] David W. Carman, Peter S. Kruus, and Brian J. Matt. "Constraints and approaches for distributed sensor network security." *NAI Labs Technical Report #00-010*, September 2000. <http://www.cs.umbc.edu/courses/graduate/CMSC691A/Spring04/papers/nailabs_report_00-010_final.pdf>.

[31] Taejoon Park and Kang G. Shin. "LiSP: A Lightweight Security Protocol for Wireless Sensor Networks." *University of Michigan.* <http://delivery.acm.org/10.1145/

1020000/1015056/p634park.pdf?key1=1015056&key2=1515136901&coll=GUIDE&dl=GUIDE&CFID=28323961&CFTOKEN=85878602>.

[32] Wendi Heinzelman. "Application Specific Protocol Architectures for Wireless Networks." *Massachusetts Institute of Technology,* June 2000.

[33] Crossbow wireless sensor network products: http://www.xbow.com/Products/Wireless_Sensor_Networks.htm

[34] Charlie Chi and Mareca Hatler, "Wireless Sensor Networks: Mass Market Opportunities" (*Market Report*), ON World Inc., Quarter 1, 2004, <http://www.onworld.com/html/wirelesssensorsrprt2.htm>.

[35] Millennial Net, Inc.: Sensor Network products, http://www.millennial.net.

[36] EmberNet Corporation: Embedded Wireless Networking Software, <http://www.ember.com>.

[37] Sensicast Systems, Inc.: Wireless Sensor Network products, <http://www.sensicast.com>.

[38] MicroStrain, Inc. Sensors and Sensor Networks products, <http://www.microstrain.com>.

[39] Bluetooth Standard, <http://www.bluetooth.org>.

[40] IEEE 802.11 Standard, 1997. <http://ieeexplore.ieee.org/iel4/5258/14251/00654749.pdf?isnumber=14251&prod=STD&arnumber=654749&arSt=i&ared=445&arAuthor=1>.

[41] David Tse and Pramod Viswanath. "Fundamentals of Wireless Communication." *Cambridge University Press,* June 2004.

[42] Lodewijk Hoesel, et al. "Prolonging the Lifetime of Wireless Sensor Networks by Cross-Layer Interaction." *University of Twente*, December 2004.

[43] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. "A Survey on Sensor Networks." *IEEE Communications Magazine,* 2002, 40(8): 102~114.

[44] K. S. J. Pister, J. M. Kahn and B. E. Boser, "Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes." *Electronics Research Laboratory*, 1999

[45] Holger Karl and Andreas Willig. "A Short Survey of Wireless Sensor Networks." *TKN Technical Report TKN-03-018*, October 2003

[46] Jane Yu and Peter Chong. "A Survey of Clustering Schemes for Mobile Ad-Hoc Networks." *Network Technology Research Center.*

[47] Newman, B. and T. Ts'o, "Kerberos: an Authentication Service for Computer Networks", *IEEE Communications Magazine*, 32 (September 1994), 33-38.

[48] Needham, R. and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM, 21(12) December 1978, 993-999.

[49] Otway D, and O. Rees, "Efficient and Timely Mutual Authentication", *Operating Systems Review*, 21 (1987), 8-10.

[50] Bellare, M. and P. Rogaway, "Entity Authentication and Key Distribution, "in *Advances in Cryptology: Proceedings of Crypto93*, LNCS 773, Springer-Verlag (1993), 232-249.

[51] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," *Proc. Hawaii Conf. System Sciences*, Jan. 2000.

[52] SunHee Yoon and Cryus Shahabi. "An Energy Conserving Clustered Aggregation Technique Leveraging Spatial Correlation," *University of Southern California.*

Appendix

A.1 Field (src/jist/swans/field/FieldInterface.java)

```
void transmit(RadioInfo srcInfo, Message msg, long duration,
double power);
```

Transmit is invoked by a node's radio to generate a transmission on the field. The field is responsible for determining what other nodes which have been registered with it are affected by the transmission, and will invoke radio.receive() and radio.endReceive() appropriately.

```
void moveRadio(Integer id, Location loc);
void moveRadioOff(Integer id, Location delta);
```

MoveRadio and moveRadio update the underlying field structure with new coordinates for the given node. This allows for node mobility, but is not utilized in the current implementation of SPECTRA.

A.2 Radio (src/jist/swans/radio/RadioInterface.java)

```
void setPower(double power);
```

SetPower adjusts the transmitter power for the next transmission. It is called by the MAC layer's function which is just a pass-through. The radio layer's setPower method is also responsible for updating the transmitPower parameter in the node's battery object.

```
void transmit(Message msg, long delay, long duration);
```

Transmit initiates a transmission. It simulates any delays required by the radio to power up the transmitter, forwards the transmit event to the field layer with the appropriate power setting, and registers the energy use with the battery. After the message transmit duration, it invokes endTransmit to clean up the radio's state.

```
void endTransmit( int sequence );
```

EndTransmit resets the radio's state to IDLE , SENSING, or RECEIVING after a transmission. it is called by radio.transmit()

```
void receive(Message msg, Double power, Long duration);
```

Receive is invoked on a radio by the field at the beginning of a transmission (after the propagation delay). The radio holds onto the message until endReceive is also called by the field. During the intervening time, the radio will flag the message with an error if it receive is called again.

```
void endReceive(Double power);
```

EndReceive is called by the field at the end of a transmission. The power value given should match the power value passed to receive. Assuming the message was transmitted without any errors, the radio forwards the message to the MAC layer.

A.3 MAC layer (src/jist/swans/MAC/MacCSMA.java)

```
void setRadioMode(byte mode);
```

SetRadioMode is called by the radio whenever the radio mode changes, to apprise the MAC layer of the change.

```
void peek(Message msg);
```

Peek is called by the radio on a receive event. Since this shows the packet to the MAC layer before the data is strictly available, peek is implemented with stub code in MacCSMA

```
void receive(Message msg, double power);
```

Receive is called by the radio when a transmission is successfully received. The MAC layer examines the destination in the MAC header and decides whether the packet is to be forwarded to the net layer. Packets may also be forwarded flagged as "promiscuous" if they are not intended for the current address. If an *ACK* is lost, the MAC layer may receive a packet twice, so packets are remembered for a period after they are acknowledged. If a duplicate packet is received, the *ACK* is resent. *ACK*s are sent immediately on reception of the packet.

```
void setRadioPower(double power);
```

SetRadioPower is called by the net layer to change the transmission strength and thus the effective radius of communication. It simply forwards the request to the radio.

```
void acknowledge( MacCSMAMessage mcm );
```

Acknowledge is called by receive to send an *ACK* to a packet.

void sendMessage(short seq, int backoffs, int noacks);

SendMessage performs the back-offs and waits for an *ACK*. If the medium is clear, it sends the message immediately; otherwise it waits for a randomized delay to resend and invokes itself. Once the packet is sent, if it needs an *ACK*, it waits for a specific delay (the acknowledge timeout), and again invokes itself. If the MAC layers acknowledge flag has been set, or the seq number has been advanced, it assumes the packet was sent successfully.

void forgetMessage(MessageID mid);

ForgetMessage is invoked by the receive event after a specific delay to clear an entry in the list of received messages.

void send(Message msg, Address nextHop);

Send is called by the network layer to send any packet other than a MAC layer acknowledge. It stores the message locally, waits for a short randomized delay (to avoid repeated contention), and invokes sendMessage

A.4 Network layer (src/jist/swans/net/NetWSN.java)

void receive(Message msg, Address lastHop, byte macId, boolean promiscuous, double rssPower_mW);

Receive is one of the most important methods in the routing layer, and is called whenever the routing layer is given a packet by the MAC layer. Receive first uses the latest *system key* to decrypt the routing header of the incoming packet. It then determines

if a given message is intended for the node that has just received it, or if that message is in a multi-hop route and needs to be forwarded to the next hop in that route.

If the message is not for the receiving node, the node will attempt to eavesdrop the packet if it is an *RREP* or data message. This eliminates the need for some later communication, and helps with data aggregation.

If the message is intended for the node, then receive decrypts the actual body of the message, and carries out an appropriate action. There is a handler for every type of message that the application can receive. For a list of message types, see section 2.7.

The message may be handed up to the application layer. The routing layer may also initiate a response without consulting the application layer, such as in the case of routing messages.

```
void send(Message msg, Address dst, short protocol, byte
priority, byte ttl);
```

Send is called by the application layer, when the application layer wants to send a message. Messages involving the application layer are generally related to data collection or aggregation. The send method is responsible for encrypting this outgoing message, and insuring that it gets a properly formatted routing header, containing a valid route to the destination. If no such route to the destination is yet known, then the message is stored, and an *RREQ* is broadcast to find a route.

```
void pump(int netid);
```

Pump is called by the routing layer to get the next message that the routing layer is waiting to transmit. Because the MAC layer can only deal with the sending of a single

message at a time, messages may be stored until the MAC layer is ready to send them. The implemented MAC layer is greedy, in that once it acquires the channel and sends, it continues to send until the sending queues are empty.

`void broadcastCHAuthenticationRequest();`

This method creates, formats, encrypts, and sends the initial *authentication request* that every CH must send to the BS as part of authentication. This message identifies the sender as a CH, and so the BS will also send a *cluster key* in response, in addition to the *system key*.

`void broadcastAuthenticationRequest();`

This method creates, formats, encrypts, and sends the initial *authentication request* that every node must send to the BS as part of authentication. This message identifies the sender as a node, and so it will receive only the *system key* in response from the BS.

`void broadcastRREQ( MessageShorts msgToSend );`

This method is called whenever any other part of the routing layer needs to establish a route to some destination. Broadcast*RREQ* encrypts the message that it receives, adds a routing header, encrypts and encapsulates the messages, and then adds it to the queue for the MAC layer, such that it will be broadcast to all nearby nodes as soon as possible.

The *RREQ* packet itself is taken as an argument to this function, and must already be correctly formatted before this method receives it. The sending nodes ID is appended to this *RREQ* message before the message is sent to the broadcast*RREQ* method.

```
void broadcastClusterJoin( MessageShorts msgToSend );
```

This message serves a dual function, in that it notifies the destination cluster that the sending node is joining the cluster, and also makes this node known to its neighbors. This message deals with encryption and adding a correct routing header.

```
void broadcastClusterAdvertisement();
```

This message is sent by CHs and informs nearby nodes that they can join its cluster. Nodes respond with a broadcastJoin message (See directly above). Nearby CHs also receive this message, and use it to populate their table of neighboring CHs, which will later be used for establishing multi-hop routes to the BS.

```
void broadcastNodeRemoval( Address.WSN toRemove );
```

This message is sent by any node that discovers that one of its neighbor nodes is missing. A node is found missing when it fails to respond with an *ACK* to a message that has been resent to it 5 times. This message instructs all receiving parties to remove that node from their neighbor tables, routing tables, and to invalidate keys associated with that node. If that node is a CH, then the receipt of this message by the BS triggers a cluster-reorganization message.

```
void joinCluster();
```

This function is called in response to the expiration of a timer that is set on system startup. This timer insures that this function will never be called until after all nodes and CHs in the system are finished authenticating themselves and broadcasting their *cluster advertisement* messages.

This function does not actually send the message to join a cluster, but rather chooses which CH to join, based on the cluster-advertisements that have been received. Once a CH has been selected, this function calls broadcastClusterJoin, which can be found above.

```
void start();
```

This method is called when the system begins, and does nothing but initiate the first attempt at authentication. It calls the broadcastAuthenticationRequest function that can be found above in this section.

```
void refreshSystemKey();
```

This method is periodically called within the routing layer of the BS, and initiates the refreshing of the *system key*. This entails generating and setting a new *system key*, encrypting the new *system key* with the old *system key*, and then broadcasting this message to all nodes in the system.

```
void unableToContactNeighbor( Address.WSN neighbor );
```

This method is called by the MAC layer, to inform the routing layer that one of the neighbors has failed to respond over 5 re-try attempts. The unableToContactNeighbor function deals with removing the missing node from tables, and formatting the *removal*

message to be sent. This *removal* message is sent by the broadcastNodeRemoval function, found above in this section.

A.5 NodeApp (src/jist/swans/app/NodeAppInterface.java)

```
void receive( Message msg, Address src );
```

Receive is called by the network layer when it receives an application packet. The application layer acts on the message based on what type it is.

```
void send_aggregated_data( short reqID );
```

CHs will schedule this event for a specified time after the first node responds to a request. It performs the aggregation on the data sent so far by the cluster's nodes and sends the result to the BS.

```
void clear_request( short reqID );
```

Send_aggregated_data schedules this event for a period of time after it sends the aggregated data. This allows the CH to disregard late responses from the nodes and helps prevent spurious transmissions. After this event, the CH has completely forgotten about the request.

```
void check_battery();
```

All nodes run check_battery in a recursive loop to monitor their energy supply. When the energy left drops below a certain threshold (currently 5% of the starting energy) it sends a message to the BS indicating that it is low on power and about to die.

```
void run();
```

The run event is called by the driver program when the node is constructed. It is responsible for kicking off the node's periodic actions and performing startup actions. There are currently no startup actions.

A.6 BSApp (src/jist/swans/app/BSAppInterface.java)

The BS app layer has some functionality in common with the node app layer. Only the features unique to the BS are described here.

```
void requestLoop();
```

Kicked off by run(), requestLoop periodically sends a request to a random CH for its aggregated data.

```
void systemKeyLoop();
```

Kicked off by run(), systemKeyLoop periodically instructs the network layer to update the *system key* in all nodes.

A.7 MAC Layer messages

The MAC layer only has two types of messages: Data messages sent by the routing layer and acknowledge packets. The MAC Header has four fields: source address, destination address, sequence number, and payload size. The source and destination addresses indicate the current hop, so one should be the current node's address, and the other should be a neighbor. The sequence number is a 16-bit counter specific to the node, so the tuple (seq, srcAddr) serves to uniquely identify a message until the counter wraps.

A.8 Net Layer Messages

ROUT_*RREQ*

Used to establish new routes in the system. This message is broadcast to all nearby nodes, and then resent by everyone who receives it until it is received by the destination. The destination then sends a *RREP* is response.

ROUT_*RREP*

Used to convey a new route back to the node that requested that route. This message is issued in response to an *RREQ* by a node that received an *RREQ* intended for it. This message is eavesdropped by all nodes who can hear it.

APP_MSG

Used primarily for data transmission and aggregation, but also used for any communication between the application layers of neighboring nodes. The contents of these messages are unknown to the routing and security layer.

APP_MSG_BCAST

Same as an APP_MSG, except that they are broadcast to all nodes, rather than directed to particular nodes.

CMD_FIRST_GROUP_KEY

This is the reply that the BS sends in response to *authentication requests*. If this message is intended for a CH, then it also contains the a new *cluster key*.

CMD_REFRESH_GROUP_KEY;

This message is periodically broadcast by the BS, and always contains a new *system key*, encrypted with the previous *system key*. This message is the primary method of continuous authentication within the SPECTRA network

CMD_NEW_CLUSTER_KEY

Sent whenever a cluster needs to distribute a new *cluster key* to it's nodes. This is not used during the *cluster organization phase*. This message is only used to refresh the *cluster key* after a node in the cluster has been compromised. The original *cluster key* during *initial system setup* is contained in the *cluster advertisement*.

CLUSTER_ADVERTISEMENT

Sent by all CHs after they are authenticated. Contains the CH ID and the *cluster key*. Nodes record the RSS with which they receive this message, and join the CH from which they received the highest RSS. This message is broadcast, and all nodes record the existence of the sending CH, for use in constructing routing tables.

CLUSTER_NODE_JOINS

Sent by all nodes after they are authenticated. Informs the CH that the node is joining the cluster. This message is broadcast, and all nodes record the existence of the sending node, for use in constructing routing tables.

SEC_NODE

This is the initial request for authentication from a node to the BS. It is broadcast, and the BS replies with a CMD_FIRST_GROUP_KEY message.

SEC_CH

This is the initial request for authentication from a CH to the BS. It is broadcast, and the BS replies with a CMD_FIRST_GROUP_KEY message.

NODE_REMOVAL

This message is broadcast when a node or CH discovers that one of it's neighbors is missing, due to node-death, wireless errors, or compromise. This message informs all recipients to remove the missing node from their routing tables, and if a CH receives this message, it will refresh its *cluster key* because the old *cluster key* may be compromised.

CLUSTER_REORG

This command is sent by the BS when it discovers that one of it's CHs is gone. The BS finds this out from a NODE_REMOVAL message. The nodes who used to

belong to the now missing CH must join nearby clusters, the same way as it would be done in node-addition, except that the cluster-less nodes already have the latest *system key*.

REFRESH_CLUSTER_KEY

Called by a CH when it discovers that one of its nodes is missing and has perhaps been compromised. This message contains a new cluster-key, encrypted with the current *system key*.

A.9 Application Layer Messages

The Application layer has the most complicated messages. Application layer messages always begin with a type which dictates the contents of the rest of the message. Depending on the message type, the second field is often a request ID, a 16-bit counter identifying the BS request that this packet is associated with and results from.

The message types, extracted from WSNApp.java, are:

```
static final short NODE_DATA = 100;
```

Data sent from a node to its CH on a periodic basis, not associated with a specific BS request.

```
static final short CH_DATA = 101;
```

Aggregated data sent from a CH to the BS, not associated with a particular request

```
static final short NODE_RESPONSE = 102;
```

Data sent from a node to its CH in response to a specific request

```
static final short CH_RESPONSE = 103;
```

Aggregated data sent from the CH to the BS in response to a specific request

```
static final short CH_REQUEST = 104;
```

Request for data from the BS to a CH

```
static final short NODE_REQUEST = 105;
```

Request for data from a CH to one or all of its nodes.

```
static final short BEGIN_MONITOR = 106;
```

Instruction for a node or CH to start making periodic reports of data

```
static final short END_MONITOR = 107;
```

Instruction for a node or CH to stop making periodic reports of data

```
static final short NODE_DEATH = 108;
```

Notification that a node is low on power